PSTAT 174 Final Project

Aliaksei Lenski

March 20, 2023

ABSTRACT

In this analysis I am investigating the Bitcoin data and how I can possibly predict the future price of Bitcoin based on the historical data. Before performing any analysis data cleaning and feature engineering is required. For the actual forecasting I am applying 2 time series models and to also compare the accuracies between both.

INTRODUCTION

Initially obtain the dataset from Kaggle. The dataset is called Bitcoin Historical Data [1]. The data was scraped using the bot that was collecting the information on the price of bitcoin every minute since Dec 31 2011 and until late 2021. Due to the unreliability of such bot alongside other multiple factors such as electricity shotdown, internet connection issues, etc., there is an enormous amout of Null values produced in the data set that needs to be dealt with. Also the time format of time stamps used needs to be changed to Day/Month/Year format to speed up the analysis.

The goal of this analysis is to attempt to create model that would be able to predict the value of Bitcoin for the next 12 months (or less). We would perform this by building 2 models: a seasonal ARIMA model, and GARCH model that is supposedly the best for high volatility financial data such as this Bitcoin dataset.

DATA EXPLORATION

The dataset essentially represents the following: CSV files for select bitcoin exchanges for the time period of Jan 2012 to December March 2021, with minute to minute updates of OHLC (Open, High, Low, Close), Volume in BTC and indicated currency, and weighted bitcoin price.

```
## [1] Starting date/time is: 1325317920
```

```
## [1] Final date/time is: 1617148800
```

```
## [1] There are 4857377 rows in our dataset
```

Let's see a couple of first rows in the dataset to interpret the schema:

```
##    Timestamp Open High  Low Close Volume_.BTC. Volume_.Currency. Weighted_Price
## 1 1325317920 4.39 4.39 4.39  4.39       0.4556                 2           4.39
## 2 1325317980  NaN  NaN  NaN   NaN          NaN               NaN            NaN
## 3 1325318040  NaN  NaN  NaN   NaN          NaN               NaN            NaN
## 4 1325318100  NaN  NaN  NaN   NaN          NaN               NaN            NaN
## 5 1325318160  NaN  NaN  NaN   NaN          NaN               NaN            NaN
## 6 1325318220  NaN  NaN  NaN   NaN          NaN               NaN            NaN
```

As we see, data cleaning and some feature engineering is required to make this dataset more analysis-friendly.

Timestamps are in Unix time [1]. Timestamps without any trades or activity have their data fields filled with NaNs. If a timestamp is missing, or if there are jumps, this may be because the exchange (or its API) was down, the exchange (or its API) did not exist, or some other unforeseen technical error in data reporting or gathering.

Let's first get rid of all the rows containing missing values and see how many rows we are going to end up with

```
## [1] After removing all the rows with missing values we now have 3613769 rows in the dataset
```

```
##    Timestamp Open High  Low Close Volume_.BTC. Volume_.Currency. Weighted_Price
## 1 1325317920 4.39 4.39 4.39  4.39       0.4556             2.000          4.390
## 2 1325346600 4.39 4.39 4.39  4.39      48.0000           210.720          4.390
## 3 1325350740 4.50 4.57 4.50  4.57      37.8623           171.380          4.526
## 4 1325350800 4.58 4.58 4.58  4.58       9.0000            41.220          4.580
## 5 1325391360 4.58 4.58 4.58  4.58       1.5020             6.879          4.580
## 6 1325431680 4.84 4.84 4.84  4.84      10.0000            48.400          4.840
```

Secondly, let's transform a Timestamp into 3 columns - Day, Month, Year; and get rid of Timestamp column:

```
##   Open High  Low Close Volume_.BTC. Volume_.Currency. Weighted_Price Day Month
## 1 4.39 4.39 4.39  4.39       0.4556             2.000          4.390  31    12
## 2 4.39 4.39 4.39  4.39      48.0000           210.720          4.390  31    12
## 3 4.50 4.57 4.50  4.57      37.8623           171.380          4.526  31    12
## 4 4.58 4.58 4.58  4.58       9.0000            41.220          4.580  31    12
## 5 4.58 4.58 4.58  4.58       1.5020             6.879          4.580  01    01
## 6 4.84 4.84 4.84  4.84      10.0000            48.400          4.840  01    01
##   Year
## 1   11
## 2   11
## 3   11
## 4   11
## 5   12
## 6   12
```

Thirdly, let's consider averaging all feature values and group by Year, Month, day because that's essentially what we need for the analysis, and even though having each day being accounted for will make the model more precise, for the purposes of this project, and computational power of my machine, Month and Year is more than enough.

```
## `summarise()` has grouped output by 'Day', 'Month'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 15 x 10
## # Groups:   Day, Month [15]
##    Day   Month Year   Open  High   Low Close VolBTC VolCurrency WeighPrice
##    <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl>  <dbl>       <dbl>      <dbl>
##  1 31    12    11     4.46  4.48  4.46  4.48 23.8        106.         4.47
##  2 01    01    12     4.81  4.81  4.81  4.81  7.20        35.3        4.81
##  3 02    01    12     5     5     5     5    19.0         95.2        5
##  4 03    01    12     5.25  5.25  5.25  5.25 11.0         58.1        5.25
```

```
##  5 04    01    12     5.2   5.22  5.2   5.22 11.9         63.1        5.21
##  6 05    01    12     6.28  6.29  6.28  6.29  4.51        28.0        6.28
##  7 06    01    12     6.44  6.44  6.44  6.44  2.42        15.9        6.44
##  8 07    01    12     6.8   6.8   6.8   6.8   0.296        2.01       6.8
##  9 08    01    12     6.95  6.95  6.95  6.95  2.5         17.3        6.95
## 10 09    01    12     6.58  6.58  6.58  6.58  1.86        12.3        6.58
## 11 10    01    12     6.60  6.60  6.60  6.60  2.08        13.8        6.60
## 12 11    01    12     7.12  7.12  7.12  7.12  2.19        15.5        7.12
## 13 12    01    12     7.06  7.06  7.06  7.06  6.86        49.8        7.06
## 14 13    01    12     6.97  6.98  6.94  6.96  1.53        10.8        6.97
## 15 14    01    12     6.42  6.42  6.41  6.41  1.68        10.8        6.41
```

```
## [1] There are 3376 rows in the final cleared dataframe
```

Let's transform the Year column into a 4-digit format and Months into abbreviated Month names

```
## # A tibble: 6 x 10
## # Groups:   Day, Month [6]
##     Day Month  Year  Open  High   Low Close VolBTC VolCurrency WeighPrice
##   <int> <chr> <int> <dbl> <dbl> <dbl> <dbl>  <dbl>       <dbl>      <dbl>
## 1    31 Dec    2011  4.46  4.48  4.46  4.48  23.8        106.        4.47
## 2     1 Jan    2012  4.81  4.81  4.81  4.81   7.20        35.3       4.81
## 3     2 Jan    2012  5     5     5     5     19.0         95.2       5
## 4     3 Jan    2012  5.25  5.25  5.25  5.25  11.0         58.1       5.25
## 5     4 Jan    2012  5.2   5.22  5.2   5.22  11.9         63.1       5.21
## 6     5 Jan    2012  6.28  6.29  6.28  6.29   4.51        28.0       6.28
```

Now that the data has been cleared and transformed, a variable to be predicted should be chosen. The most reasonable and obvious choice would be the Weighted_Price or VWAP- Volume Weighted Average Price. It is a clear indication on the change of Bitcoin's value that would be interesting to predict.

I decided to remove the very first Observation which is December 31, 2011 to make a matrix clearer and easier to comprehend

```
## # A tibble: 15 x 4
## # Groups:   Day, Month [15]
##    Month   Day  Year WeighPrice
##    <chr> <int> <int>      <dbl>
##  1 Jan       1  2012       4.81
##  2 Jan       2  2012       5
##  3 Jan       3  2012       5.25
##  4 Jan       4  2012       5.21
##  5 Jan       5  2012       6.28
##  6 Jan       6  2012       6.44
##  7 Jan       7  2012       6.8
##  8 Jan       8  2012       6.95
##  9 Jan       9  2012       6.58
## 10 Jan      10  2012       6.60
## 11 Jan      11  2012       7.12
## 12 Jan      12  2012       7.06
## 13 Jan      13  2012       6.97
## 14 Jan      14  2012       6.41
## 15 Jan      15  2012       7.15
```

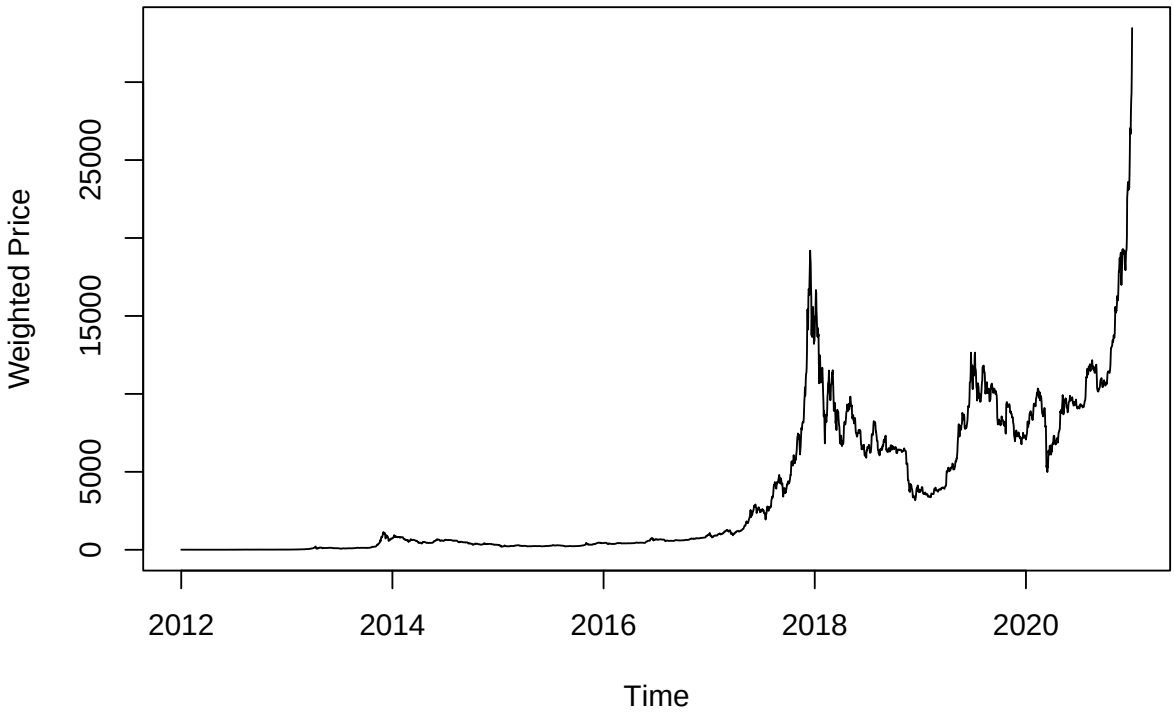
Now that the data is ready for the Analysis, we can transform it into a Time Series data and start the analysis

TIME SERIES DATA

First thing to do would be to transform the dataset into time series data, we would make it a daily time series

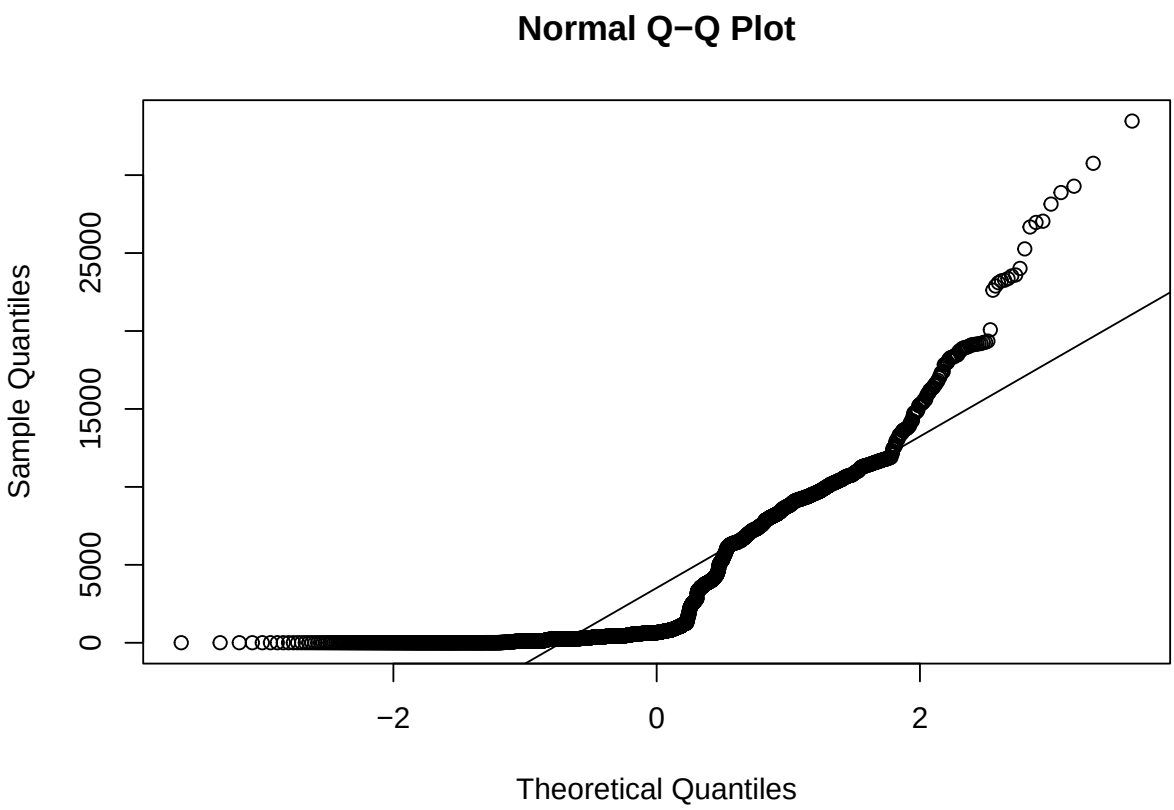
```
## Time Series:
## Start = c(2012, 1)
## End = c(2012, 6)
## Frequency = 365
## [[1]]
## [1] 4.807
##
## [[2]]
## [1] 5
##
## [[3]]
## [1] 5.252
##
## [[4]]
## [1] 5.208
##
## [[5]]
## [1] 6.284
##
## [[6]]
## [1] 6.439
```

Now that I made a matrix of the Weighted Price per day, now let's see how this data would look like on a time series plot



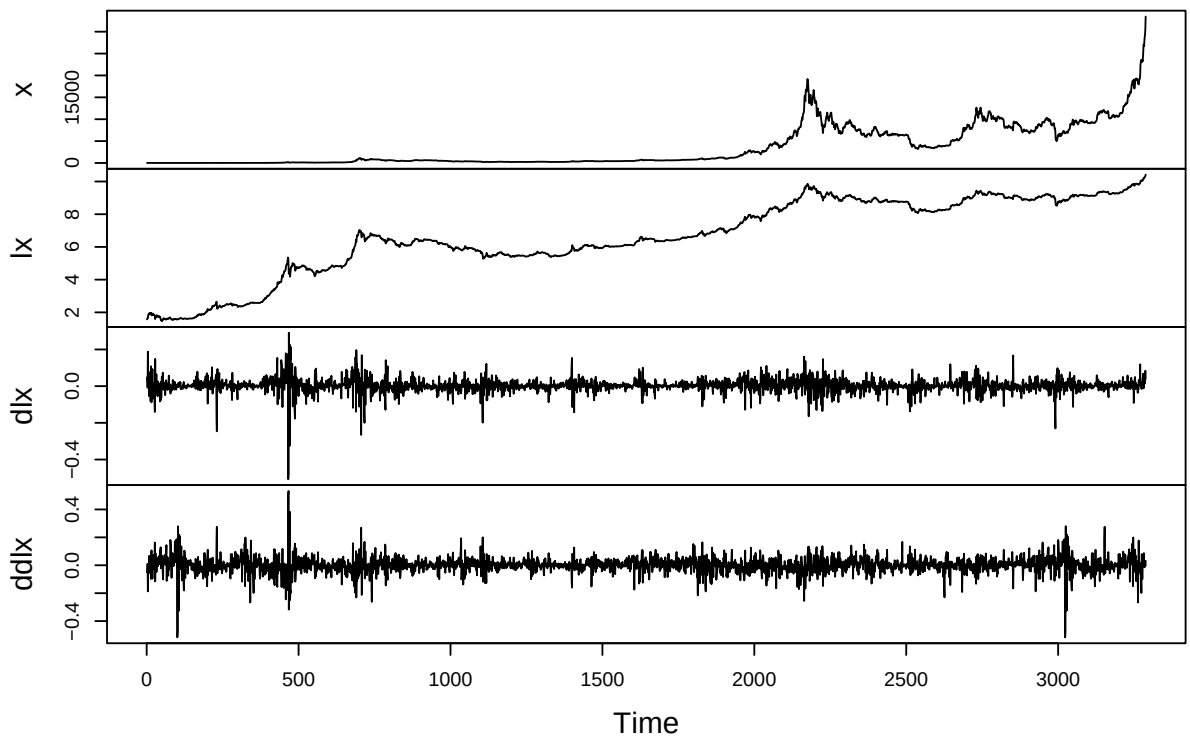
Data doesn't look stationary some transformations are certainly needed.

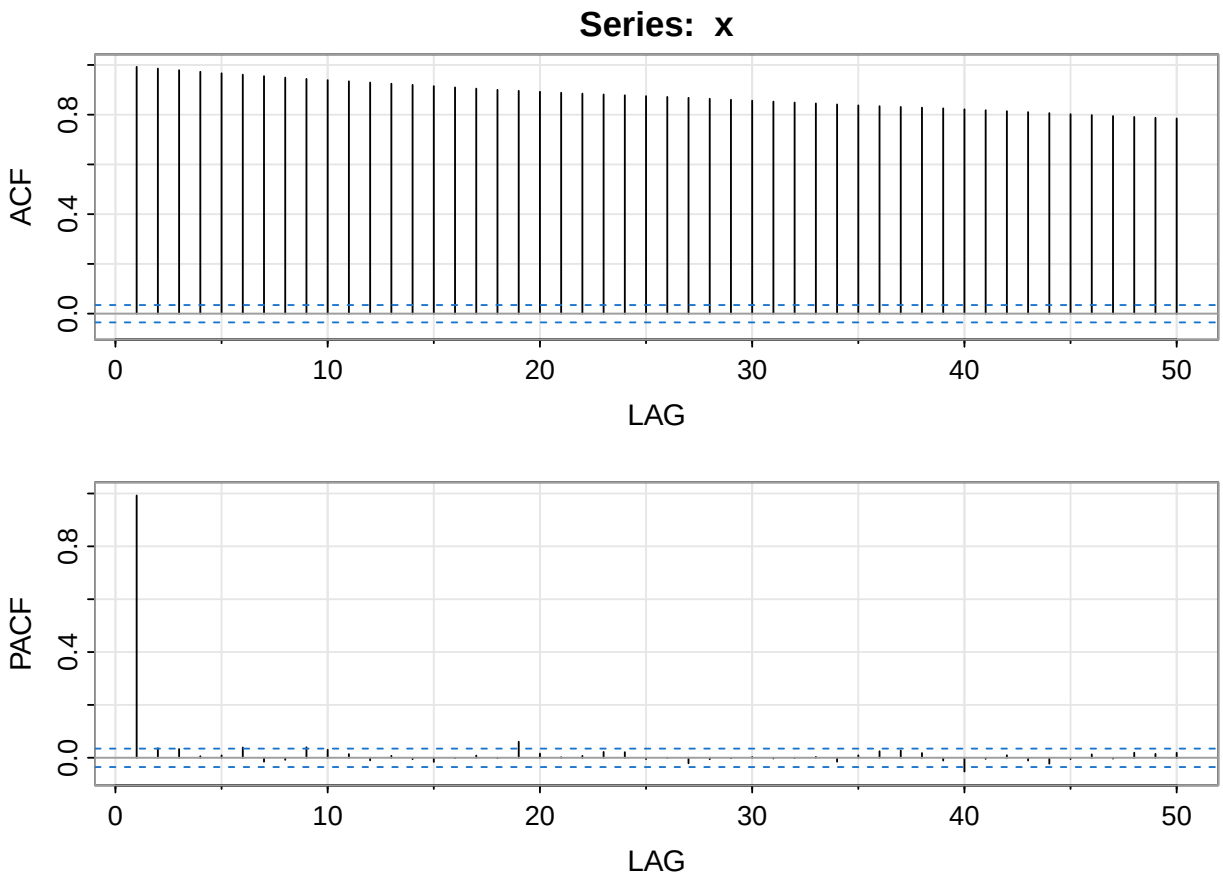
Let's construct a QQ-plot on the data to see if the given data is normal



The data is clearly not normal. The log transformation is needed [2]. Let's transform the data in order to make it more stationary:

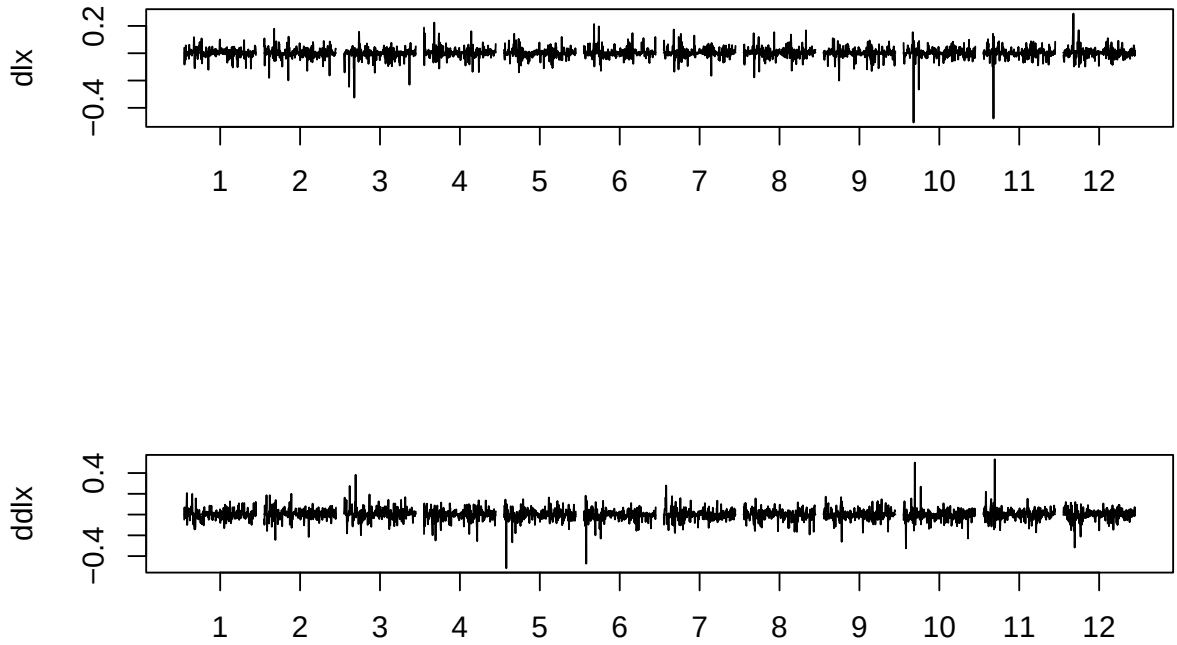
```
## Warning in cbind(x, lx, dlx, ddlx): number of rows of result is not a multiple
## of vector length (arg 3)
```


cbind(x, lx, dlx, ddlx)
x
lx
dlx
ddlx
0
15000
2
4
6
8
−0.4
0.0
0.4
0
500
1000
1500
2000
2500
3000
Time

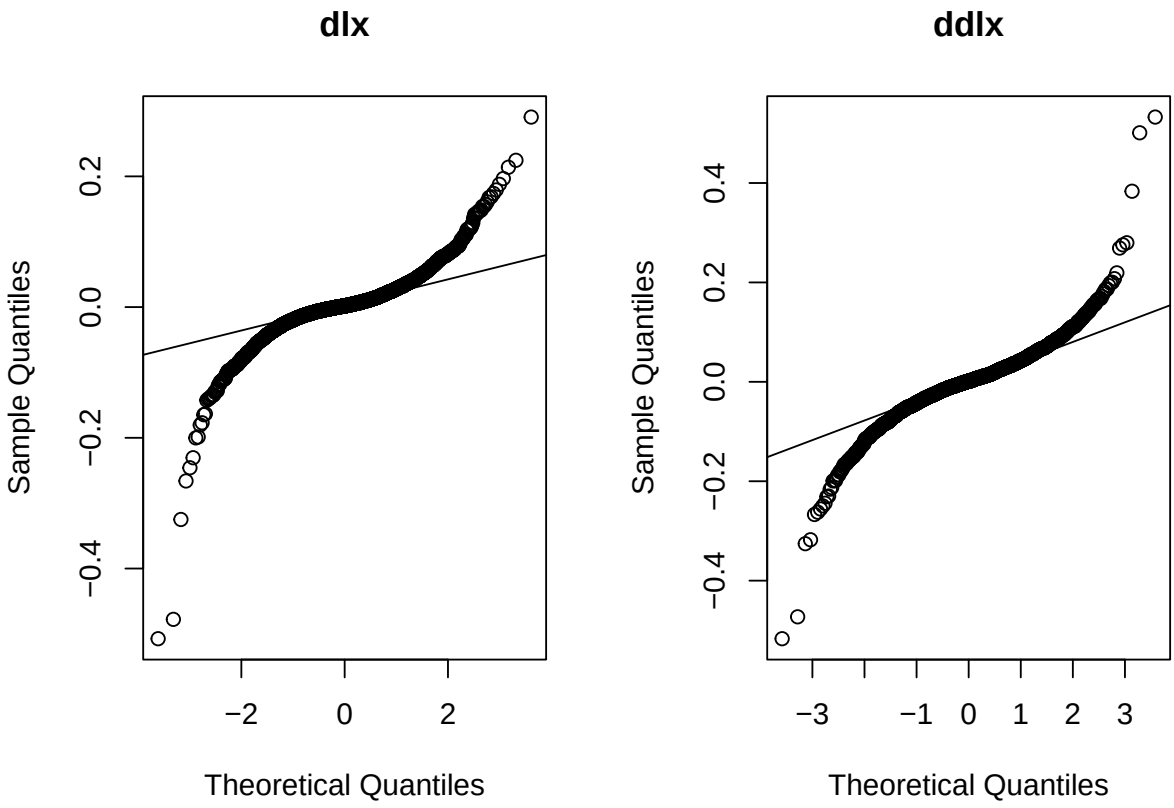



```
##      [,1] [,2] [,3] [,4] [,5] [,6]  [,7]  [,8] [,9] [,10] [,11] [,12] [,13]
## ACF  0.99 0.99 0.98 0.97 0.97 0.96  0.96  0.95 0.94  0.94  0.93  0.93  0.92
## PACF 0.99 0.04 0.03 0.01 0.01 0.04 -0.02 -0.01 0.04  0.03  0.01 -0.01  0.01
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF   0.92  0.91  0.91  0.90   0.9  0.90  0.89  0.89  0.88  0.88  0.88  0.87
## PACF -0.01 -0.02  0.00  0.01   0.0  0.06  0.02  0.00  0.01  0.02  0.02 -0.01
##      [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37]
## ACF   0.87  0.87  0.86  0.86  0.86  0.85  0.85  0.85  0.84  0.84  0.83  0.83
## PACF  0.00 -0.02 -0.01  0.00  0.00  0.00  0.00  0.00 -0.02  0.01  0.02  0.03
##      [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49]
## ACF   0.83  0.83  0.82  0.82  0.81  0.81  0.81  0.80  0.80  0.79  0.79  0.79
## PACF  0.02 -0.01 -0.05  0.00  0.01 -0.01 -0.02 -0.01  0.01  0.00  0.02  0.01
##      [,50]
## ACF   0.78
## PACF  0.02
```

I observed that from all the tried transformations on the data the most stationary looking ones are dlx and ddlx, basically ARIMA with d=1 and ARIMA with d=2 respectively [2]. Let's compare their monthly plot to see which one looks more stationary than the other.



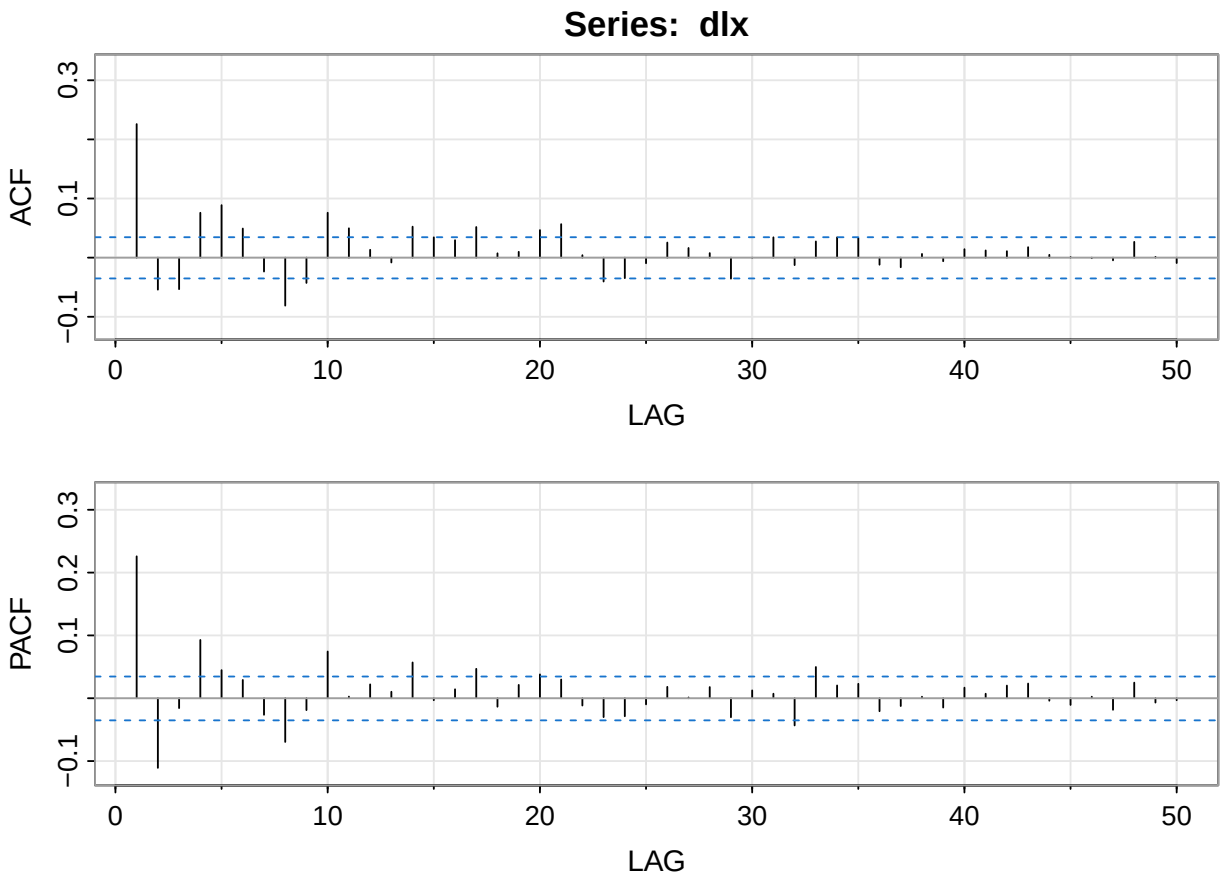
This test doesn't seem to work for this dataset. It's hard to determine by these graphs which is the best model. Let's choose the best fit by comparing their variances and making qqplots for both. If one has a better fit into a qqline we will choose it, if the difference is insignificant, we will choose the one with smaller variance




```
## [1] The variance of dlx:  0.00148722335508483
```

```
## [1] The variance of ddlx:  0.0032306727633523
```

Judging by the qqplots for both, ddlx might have a better normality, but the difference is insignificant. Only once differentiated data has the smallest variance so I will use d=1 on the ARIMA[2].




```
##      [,1]  [,2]  [,3] [,4] [,5] [,6]  [,7]  [,8]  [,9] [,10] [,11] [,12] [,13]
## ACF  0.23 -0.05 -0.05 0.08 0.09 0.05 -0.02 -0.08 -0.04  0.08  0.05  0.01 -0.01
## PACF 0.23 -0.11 -0.02 0.09 0.04 0.03 -0.03 -0.07 -0.02  0.07  0.00  0.02  0.01
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF   0.05  0.03  0.03  0.05  0.01  0.01  0.05  0.06  0.00 -0.04 -0.03 -0.01
## PACF  0.06  0.00  0.01  0.05 -0.01  0.02  0.04  0.03 -0.01 -0.03 -0.03 -0.01
##      [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37]
## ACF   0.03  0.02  0.01 -0.04  0.00  0.03 -0.01  0.03  0.03  0.03 -0.01 -0.02
## PACF  0.02  0.00  0.02 -0.03  0.01  0.01 -0.04  0.05  0.02  0.02 -0.02 -0.01
##      [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49]
## ACF   0.01 -0.01  0.01  0.01  0.01  0.02     0  0.00     0  0.00  0.03  0.00
## PACF  0.00 -0.01  0.02  0.01  0.02  0.02     0 -0.01     0 -0.02  0.02 -0.01
##      [,50]
## ACF  -0.01
## PACF  0.00
```

FITTING BEST ARIMA MODEL

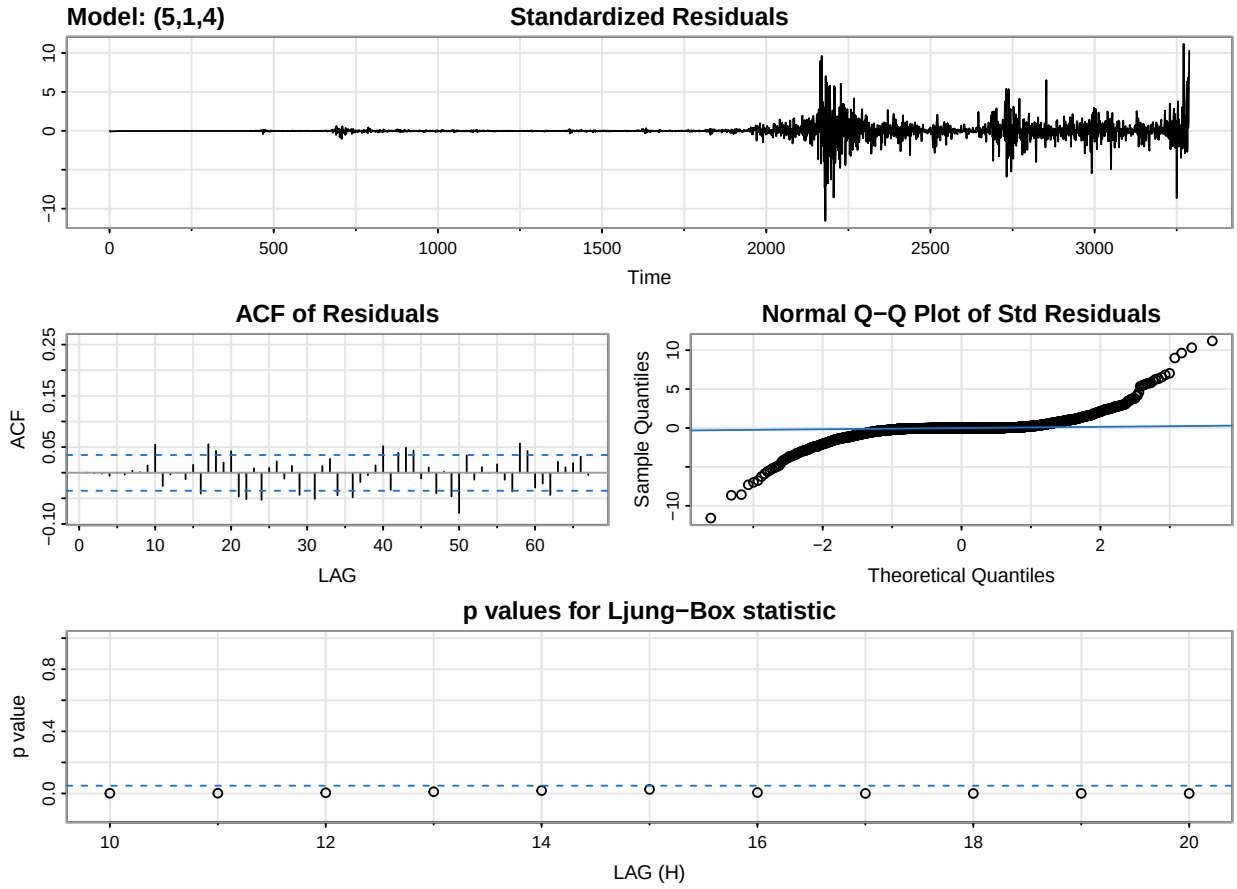
```
## Series: x
## ARIMA(1,2,4)
##
## Coefficients:
##          ar1    ma1     ma2     ma3   ma4
##       -0.918  0.191  -0.906  -0.248  0.03
```

```
## s.e.   0.040  0.044   0.032   0.018  0.02
##
## sigma^2 = 42522:  log likelihood = -22172
## AIC=44356   AICc=44356   BIC=44393
```

By choosing the values of ARIMA model based on the ACF and PACF and taking auto.arima into consideration, I decided to use the following model: ARIMA(5 , 1, 4) since it looks more aligning to the data than auto.arima [2].

```
## Warning in sqrt(diag(fitit$var.coef)): NaNs produced
```

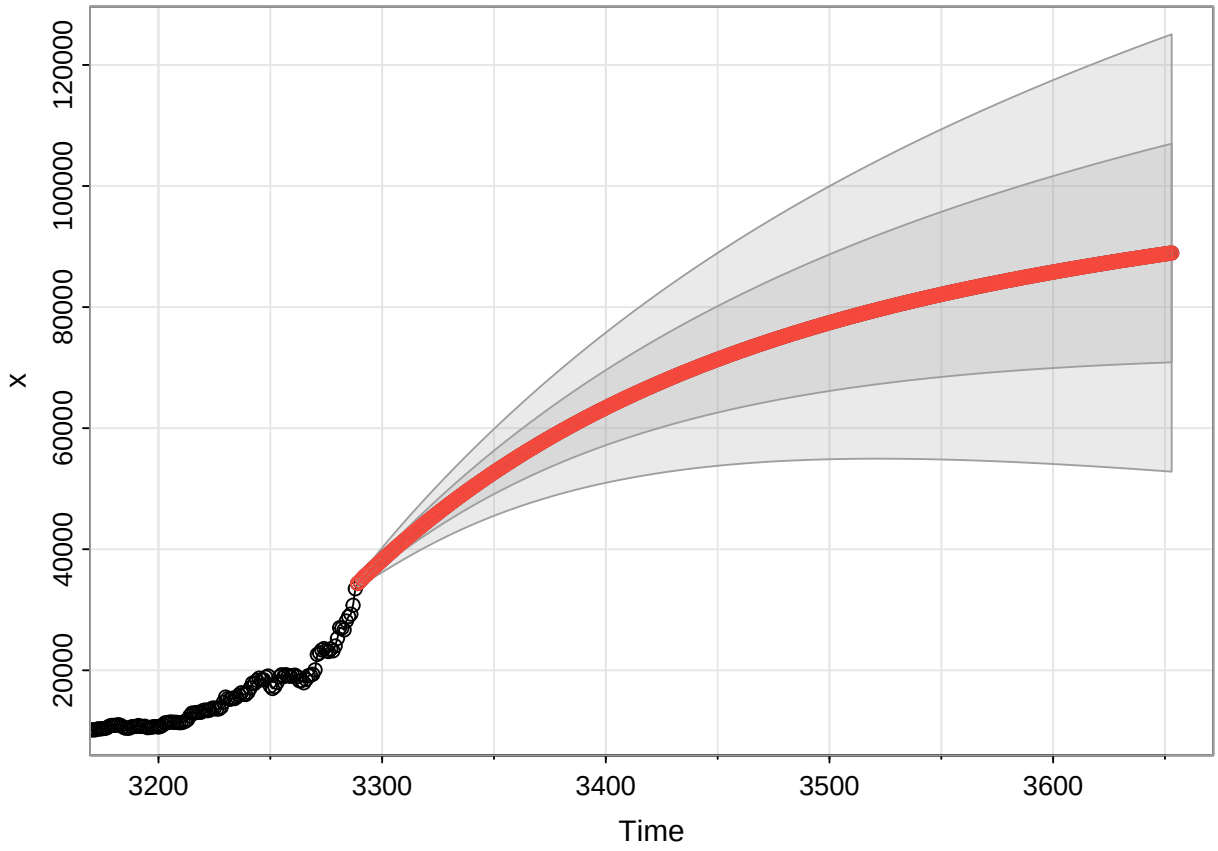
```
## Warning in sqrt(diag(fitit$var.coef)): NaNs produced
```



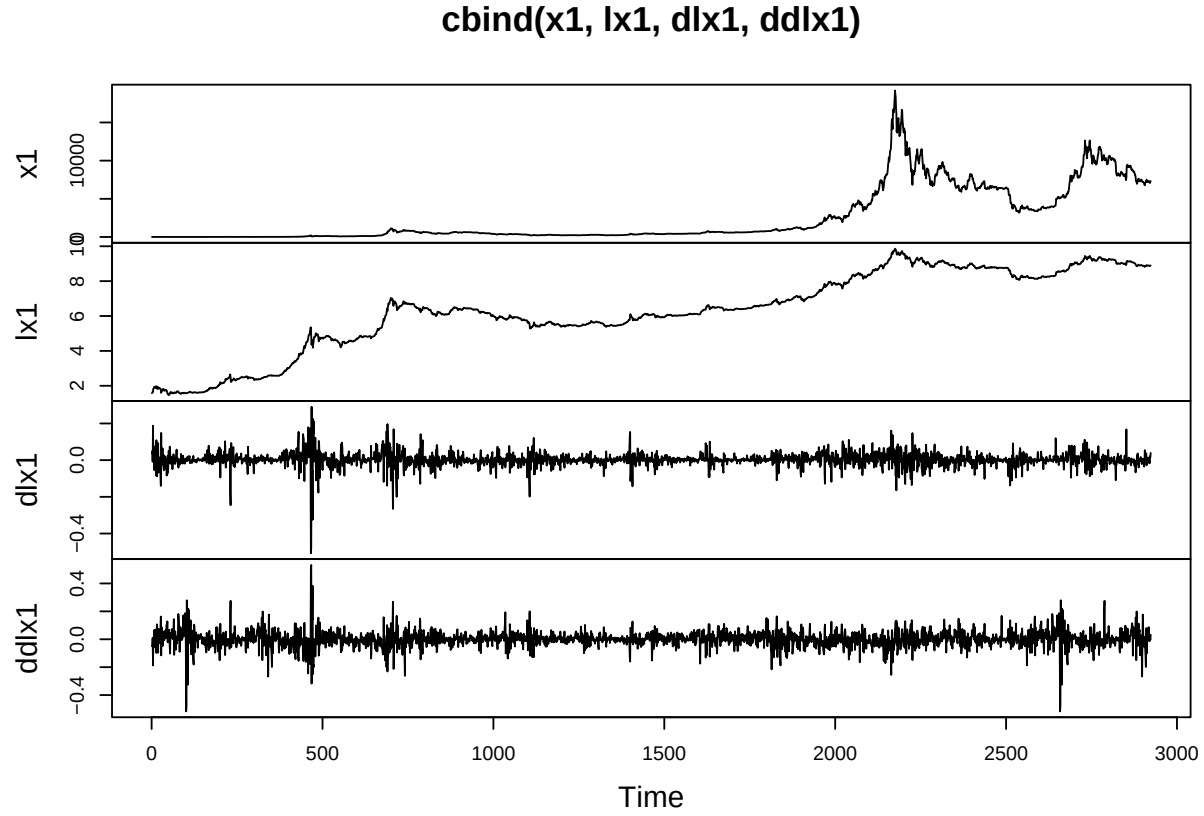

```
## Warning in sqrt(diag(x$var.coef)): NaNs produced
```

Having all p-values below the margin for any choice of p and q parameters suggests that the model doesnt fit well in the dataset.

For forecasting I decided to use ARIMA(10,1,11) or SARIMA(5,1,4)x(0,0,0) and forecast the next 365 days

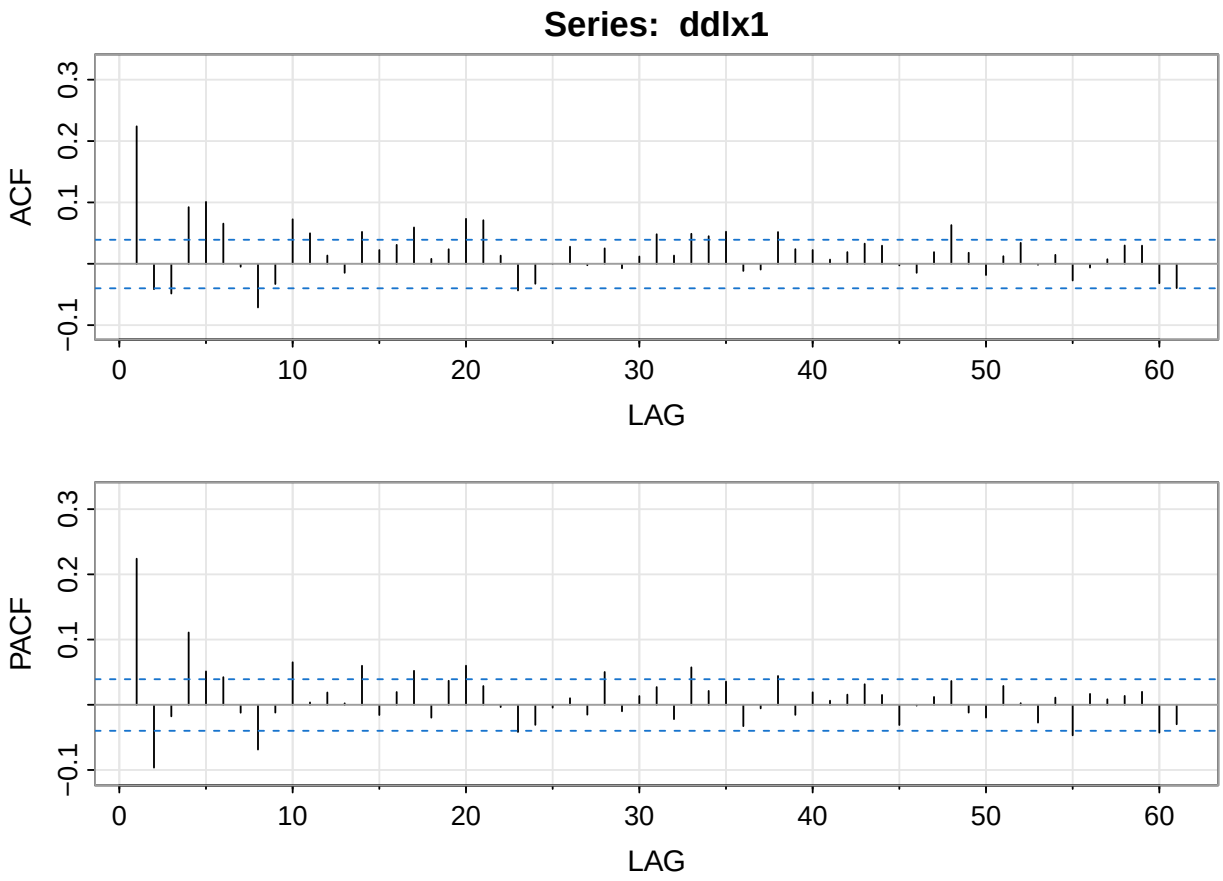


From the p-values and qqplot, ARIMA model doesn't look too reliable to accurately predict the weighted average price of bitcoin. Let's attempt one more check before choosing another model and create a new time series variable with the end date a year before the actual end date and see how well would it fit.




```
## [1] 0.001562
```

```
## [1] 0.003438
```

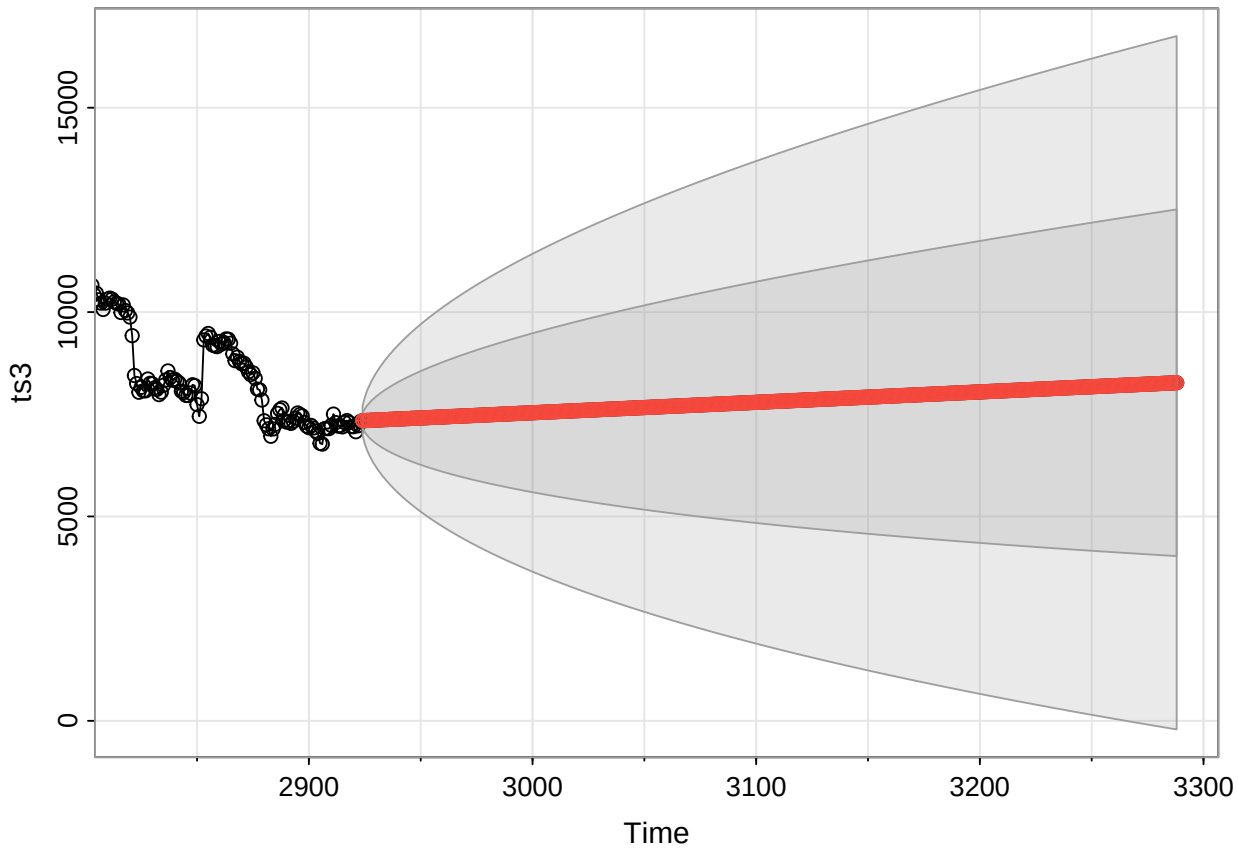



```
##      [,1]  [,2]  [,3] [,4] [,5] [,6]  [,7]  [,8]  [,9] [,10] [,11] [,12] [,13]
## ACF  0.22 -0.04 -0.05 0.09 0.10 0.07  0.00 -0.07 -0.03  0.07  0.05  0.01 -0.01
## PACF 0.22 -0.10 -0.02 0.11 0.05 0.04 -0.01 -0.07 -0.01  0.07  0.00  0.02  0.00
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF   0.05  0.02  0.03  0.06  0.01  0.02  0.07  0.07  0.01 -0.04 -0.03     0
## PACF  0.06 -0.02  0.02  0.05 -0.02  0.04  0.06  0.03  0.00 -0.04 -0.03     0
##      [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37]
## ACF   0.03  0.00  0.03 -0.01  0.01  0.05  0.01  0.05  0.04  0.05 -0.01 -0.01
## PACF  0.01 -0.02  0.05 -0.01  0.01  0.03 -0.02  0.06  0.02  0.04 -0.03 -0.01
##      [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49]
## ACF   0.05  0.02  0.02  0.01  0.02  0.03  0.03  0.00 -0.01  0.02  0.06  0.02
## PACF  0.04 -0.02  0.02  0.01  0.02  0.03  0.02 -0.03  0.00  0.01  0.04 -0.01
##      [,50] [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61]
## ACF  -0.02  0.01  0.03  0.00  0.01 -0.03 -0.01  0.01  0.03  0.03 -0.03 -0.04
## PACF -0.02  0.03  0.00 -0.03  0.01 -0.05  0.02  0.01  0.01  0.02 -0.04 -0.03
```

So, based on the variance comparison and ACF & PACF graphs, for this data the most appropriate model seems to be ARIMA(1,1,2) or SARIMA(1,1,2)x(0,0,0) and forecast the next 365 days.

```
## Series: x1
## ARIMA(1,1,2)
##
## Coefficients:
##          ar1     ma1     ma2
```

```
##       -0.934  1.143  0.230
## s.e.   0.024  0.029  0.018
##
## sigma^2 = 32796:  log likelihood = -19336
## AIC=38681   AICc=38681   BIC=38704
```




```
## $pred
## Time Series:
## Start = 2924
## End = 3288
## Frequency = 1
##   [1] 7346 7344 7351 7349 7356 7355 7360 7360 7365 7365 7370 7370 7375 7376 7380
##  [16] 7381 7385 7386 7390 7391 7395 7396 7400 7402 7405 7407 7410 7412 7415 7417
##  [31] 7420 7422 7425 7427 7430 7432 7435 7438 7441 7443 7446 7448 7451 7453 7456
##  [46] 7458 7461 7463 7466 7468 7471 7473 7476 7479 7481 7484 7486 7489 7491 7494
##  [61] 7496 7499 7502 7504 7507 7509 7512 7514 7517 7519 7522 7524 7527 7530 7532
##  [76] 7535 7537 7540 7542 7545 7547 7550 7552 7555 7558 7560 7563 7565 7568 7570
##  [91] 7573 7575 7578 7581 7583 7586 7588 7591 7593 7596 7598 7601 7603 7606 7609
## [106] 7611 7614 7616 7619 7621 7624 7626 7629 7631 7634 7637 7639 7642 7644 7647
## [121] 7649 7652 7654 7657 7659 7662 7665 7667 7670 7672 7675 7677 7680 7682 7685
## [136] 7688 7690 7693 7695 7698 7700 7703 7705 7708 7710 7713 7716 7718 7721 7723
## [151] 7726 7728 7731 7733 7736 7738 7741 7744 7746 7749 7751 7754 7756 7759 7761
## [166] 7764 7767 7769 7772 7774 7777 7779 7782 7784 7787 7789 7792 7795 7797 7800
## [181] 7802 7805 7807 7810 7812 7815 7817 7820 7823 7825 7828 7830 7833 7835 7838
## [196] 7840 7843 7845 7848 7851 7853 7856 7858 7861 7863 7866 7868 7871 7874 7876
```

```
## [211] 7879 7881 7884 7886 7889 7891 7894 7896 7899 7902 7904 7907 7909 7912 7914
## [226] 7917 7919 7922 7924 7927 7930 7932 7935 7937 7940 7942 7945 7947 7950 7953
## [241] 7955 7958 7960 7963 7965 7968 7970 7973 7975 7978 7981 7983 7986 7988 7991
## [256] 7993 7996 7998 8001 8003 8006 8009 8011 8014 8016 8019 8021 8024 8026 8029
## [271] 8032 8034 8037 8039 8042 8044 8047 8049 8052 8054 8057 8060 8062 8065 8067
## [286] 8070 8072 8075 8077 8080 8082 8085 8088 8090 8093 8095 8098 8100 8103 8105
## [301] 8108 8110 8113 8116 8118 8121 8123 8126 8128 8131 8133 8136 8139 8141 8144
## [316] 8146 8149 8151 8154 8156 8159 8161 8164 8167 8169 8172 8174 8177 8179 8182
## [331] 8184 8187 8189 8192 8195 8197 8200 8202 8205 8207 8210 8212 8215 8218 8220
## [346] 8223 8225 8228 8230 8233 8235 8238 8240 8243 8246 8248 8251 8253 8256 8258
## [361] 8261 8263 8266 8268 8271
##
## $se
## Time Series:
## Start = 2924
## End = 3288
## Frequency = 1
##   [1]  181.0  283.9  362.4  423.5  479.5  527.3  573.1  613.8  653.5  689.6
##  [11]  725.0  757.8  790.1  820.3  850.2  878.4  906.3  932.9  959.2  984.3
##  [21] 1009.3 1033.2 1057.0 1079.9 1102.7 1124.7 1146.5 1167.7 1188.7 1209.2
##  [31] 1229.5 1249.3 1269.0 1288.2 1307.3 1325.9 1344.5 1362.6 1380.6 1398.4
##  [41] 1415.9 1433.2 1450.3 1467.2 1483.9 1500.4 1516.8 1533.0 1549.0 1564.8
##  [51] 1580.5 1596.0 1611.4 1626.6 1641.7 1656.7 1671.5 1686.2 1700.7 1715.2
##  [61] 1729.5 1743.7 1757.8 1771.7 1785.6 1799.4 1813.0 1826.6 1840.0 1853.4
##  [71] 1866.6 1879.8 1892.9 1905.8 1918.7 1931.5 1944.3 1956.9 1969.5 1982.0
##  [81] 1994.4 2006.7 2018.9 2031.1 2043.2 2055.2 2067.2 2079.1 2090.9 2102.7
##  [91] 2114.4 2126.0 2137.6 2149.1 2160.5 2171.9 2183.2 2194.5 2205.7 2216.9
## [101] 2228.0 2239.0 2250.0 2260.9 2271.8 2282.6 2293.4 2304.1 2314.8 2325.4
## [111] 2336.0 2346.5 2357.0 2367.5 2377.9 2388.2 2398.5 2408.8 2419.0 2429.2
## [121] 2439.3 2449.4 2459.4 2469.4 2479.4 2489.3 2499.2 2509.1 2518.9 2528.6
## [131] 2538.4 2548.1 2557.7 2567.3 2576.9 2586.5 2596.0 2605.5 2614.9 2624.3
## [141] 2633.7 2643.1 2652.4 2661.7 2670.9 2680.1 2689.3 2698.5 2707.6 2716.7
## [151] 2725.7 2734.8 2743.8 2752.7 2761.7 2770.6 2779.5 2788.3 2797.2 2806.0
## [161] 2814.7 2823.5 2832.2 2840.9 2849.6 2858.2 2866.8 2875.4 2884.0 2892.5
## [171] 2901.0 2909.5 2918.0 2926.4 2934.8 2943.2 2951.6 2959.9 2968.3 2976.5
## [181] 2984.8 2993.1 3001.3 3009.5 3017.7 3025.8 3034.0 3042.1 3050.2 3058.3
## [191] 3066.3 3074.4 3082.4 3090.4 3098.3 3106.3 3114.2 3122.1 3130.0 3137.9
## [201] 3145.7 3153.5 3161.3 3169.1 3176.9 3184.7 3192.4 3200.1 3207.8 3215.5
## [211] 3223.1 3230.8 3238.4 3246.0 3253.6 3261.2 3268.7 3276.3 3283.8 3291.3
## [221] 3298.8 3306.2 3313.7 3321.1 3328.5 3335.9 3343.3 3350.7 3358.0 3365.4
## [231] 3372.7 3380.0 3387.3 3394.5 3401.8 3409.0 3416.3 3423.5 3430.7 3437.9
## [241] 3445.0 3452.2 3459.3 3466.4 3473.5 3480.6 3487.7 3494.8 3501.8 3508.8
## [251] 3515.9 3522.9 3529.9 3536.8 3543.8 3550.8 3557.7 3564.6 3571.5 3578.4
## [261] 3585.3 3592.2 3599.0 3605.9 3612.7 3619.5 3626.4 3633.1 3639.9 3646.7
## [271] 3653.4 3660.2 3666.9 3673.6 3680.3 3687.0 3693.7 3700.4 3707.1 3713.7
## [281] 3720.3 3727.0 3733.6 3740.2 3746.7 3753.3 3759.9 3766.4 3773.0 3779.5
## [291] 3786.0 3792.5 3799.0 3805.5 3812.0 3818.5 3824.9 3831.4 3837.8 3844.2
## [301] 3850.6 3857.0 3863.4 3869.8 3876.1 3882.5 3888.8 3895.2 3901.5 3907.8
## [311] 3914.1 3920.4 3926.7 3933.0 3939.2 3945.5 3951.7 3958.0 3964.2 3970.4
## [321] 3976.6 3982.8 3989.0 3995.2 4001.4 4007.5 4013.7 4019.8 4025.9 4032.0
## [331] 4038.2 4044.3 4050.4 4056.4 4062.5 4068.6 4074.6 4080.7 4086.7 4092.7
## [341] 4098.8 4104.8 4110.8 4116.8 4122.8 4128.7 4134.7 4140.7 4146.6 4152.6
## [351] 4158.5 4164.4 4170.3 4176.2 4182.1 4188.0 4193.9 4199.8 4205.7 4211.5
## [361] 4217.4 4223.2 4229.0 4234.9 4240.7
```

As we observed here a simple ARIMA model can't accurately predict this data, so we should discard this very general model go for something different and more complex.

FITTING GARCH MODEL

Since I am working with a financial time series data, we would like to model volatility of the average price of bitcoin and later use this model for forecasting [2].

It also might be a better fit than the original ARIMA/SARIMA model because it takes into account the changes in volatility over time. So, as we observed our data previously, the variance in price changes quite drastically in 2018 and 2022 in comparisson to the previous years, so GARCH model can be more accurate than seasonal ARIMA for our data.

Since GARCH models assume that the data they are being applied to is stationary, I would use the transformed data dlx (since it has been decided to be the best transformation) and make it a time series data to which I will fit the model later [2]

```
## Warning in adf.test(ts_dlx): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_dlx
## Dickey-Fuller = -12, Lag order = 14, p-value = 0.01
## alternative hypothesis: stationary
```

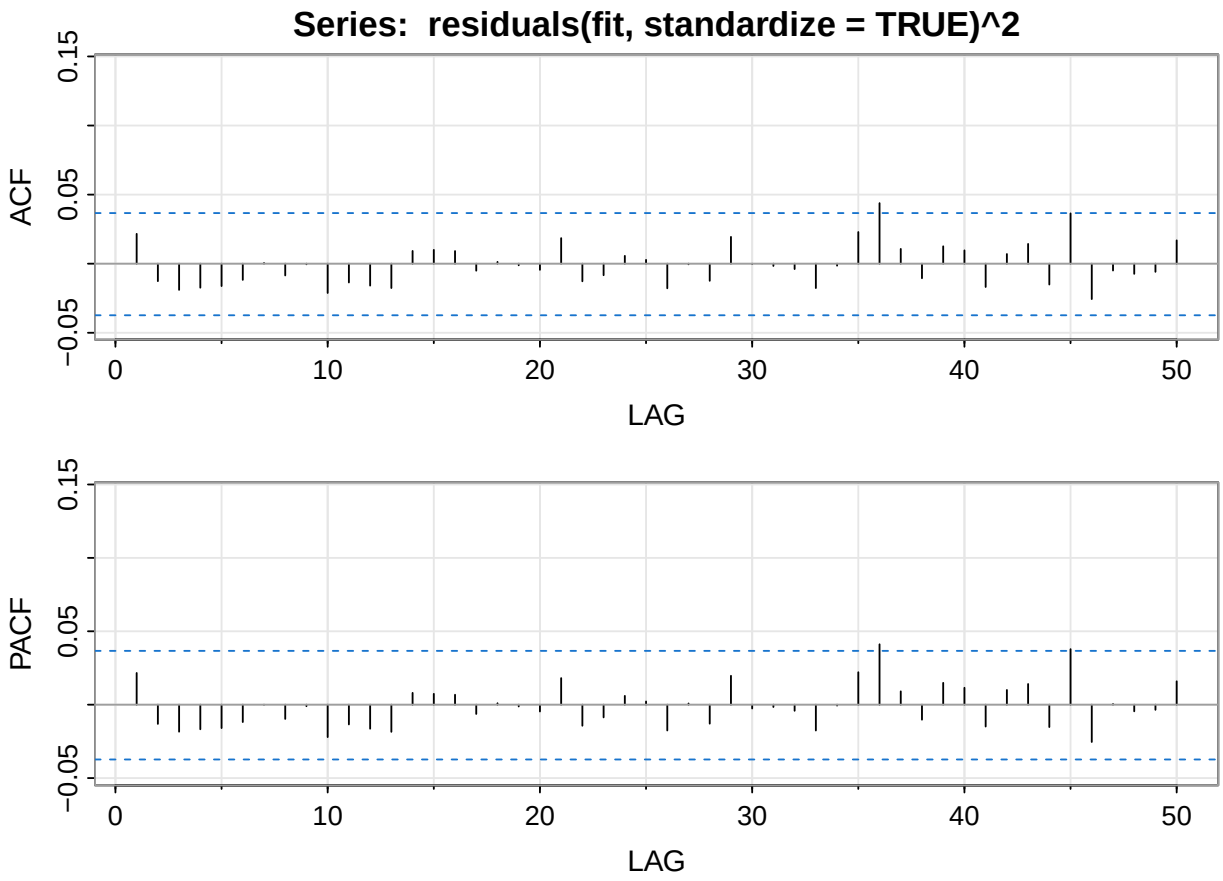
Having p-value so low, we can assume the data is stationary [2]. We can start fitting the model.

Let's check the residuals for model normality and autocorrelation assumptions:

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = ~arma(1, 1) + garch(1, 1), data = ts_dlx,
##     trace = FALSE)
##
## Mean and Variance Equation:
##  data ~ arma(1, 1) + garch(1, 1)
## <environment: 0x000001659ba2ca40>
##  [data = ts_dlx]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##          mu          ar1          ma1        omega       alpha1        beta1
##  1.4030e-03  -2.3446e-02   3.2247e-01   3.0849e-05   2.6984e-01   7.4193e-01
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
```

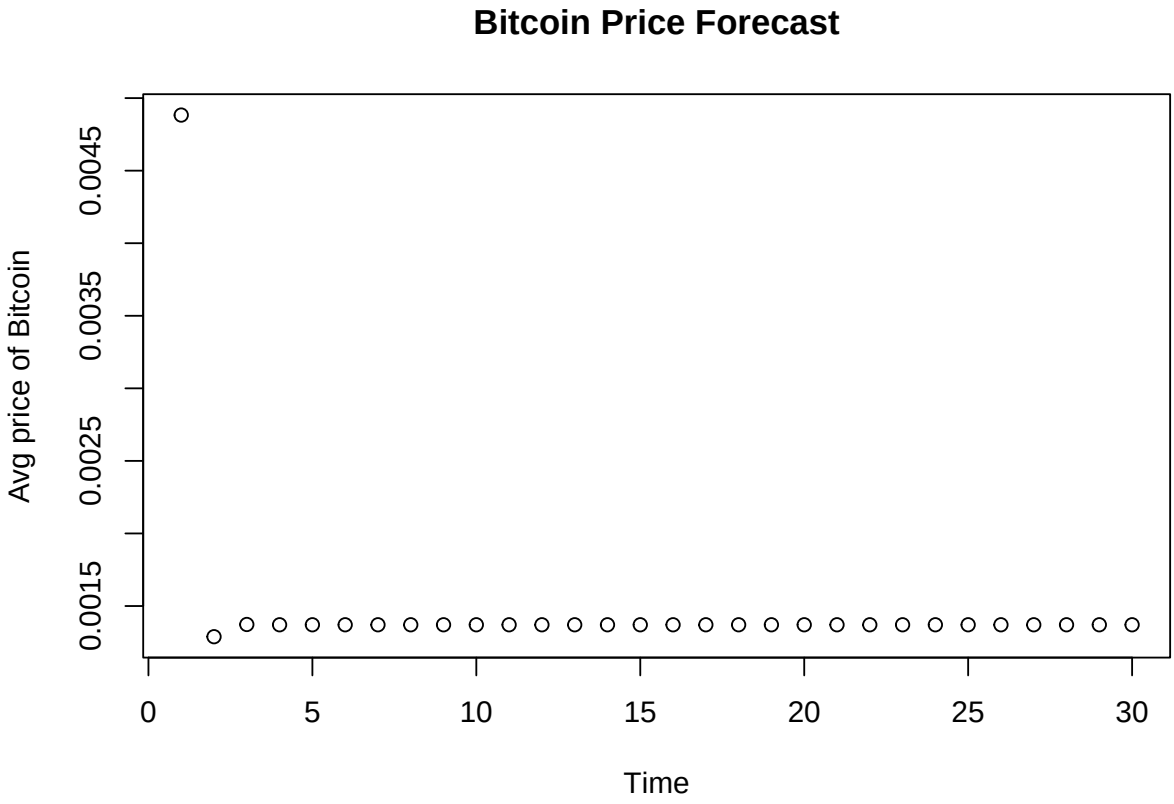
```
##          Estimate  Std. Error  t value Pr(>|t|)
## mu      1.403e-03   5.279e-04    2.658  0.00787 **
## ar1    -2.345e-02   6.906e-02   -0.339  0.73425
## ma1     3.225e-01   6.514e-02    4.950 7.41e-07 ***
## omega   3.085e-05   4.522e-06    6.822 8.96e-12 ***
## alpha1  2.698e-01   2.195e-02   12.295  < 2e-16 ***
## beta1   7.419e-01   1.684e-02   44.058  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  6336    normalized:  2.168
##
## Description:
##  Mon Mar 20 14:11:15 2023 by user: alexe
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  2018      0
##  Shapiro-Wilk Test  R    W      0.9581    0
##  Ljung-Box Test     R    Q(10)  60        3.624e-09
##  Ljung-Box Test     R    Q(15)  76.94     2.517e-10
##  Ljung-Box Test     R    Q(20)  90.45     6.189e-11
##  Ljung-Box Test     R^2  Q(10)  6.477     0.7738
##  Ljung-Box Test     R^2  Q(15)  9.218     0.8659
##  Ljung-Box Test     R^2  Q(20)  9.608     0.9747
##  LM Arch Test       R    TR^2   8.229     0.767
##
## Information Criterion Statistics:
##    AIC    BIC    SIC   HQIC
## -4.331 -4.319 -4.331 -4.327
```

From the summary output, the p-values and AIC&BIC, I can make a conclusion that GARCH might be a good fit to forecast the data [2].




```
##      [,1]  [,2]  [,3]  [,4]  [,5]  [,6] [,7]  [,8] [,9] [,10] [,11] [,12] [,13]
## ACF  0.02 -0.01 -0.02 -0.02 -0.02 -0.01    0 -0.01    0 -0.02 -0.01 -0.02 -0.02
## PACF 0.02 -0.01 -0.02 -0.02 -0.02 -0.01    0 -0.01    0 -0.02 -0.01 -0.02 -0.02
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF   0.01  0.01  0.01 -0.01     0     0     0  0.02 -0.01 -0.01  0.01     0
## PACF  0.01  0.01  0.01 -0.01     0     0     0  0.02 -0.01 -0.01  0.01     0
##      [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37]
## ACF  -0.02     0 -0.01  0.02     0     0     0 -0.02     0  0.02  0.04  0.01
## PACF -0.02     0 -0.01  0.02     0     0     0 -0.02     0  0.02  0.04  0.01
##      [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49]
## ACF  -0.01  0.01  0.01 -0.02  0.01  0.01 -0.02  0.04 -0.03     0 -0.01 -0.01
## PACF -0.01  0.01  0.01 -0.01  0.01  0.01 -0.02  0.04 -0.03     0  0.00  0.00
##      [,50]
## ACF   0.02
## PACF  0.02
```

I decided, that it might be a little difficult to predict the next 365 values using GARCH, so instead I chose to forecast the next 30 days to see a more realistic prediction.



This graph shows that for the next day the average bitcoin price is still going to be relatively high, but after that it will drop massively and stay on a similar low average price for the entire predicted 30 days.

Now that we have our prediction, let's check the accuracy:

1) Mean Absolute Error:

```
## [1] 0.01343
```

2) Mean Squared Error:

```
## [1] 0.0003177
```

Both Errors are pretty low, which would suggest relatively high accuracy of the model [2]

CONCLUSIONS

In this project was observed the behavior of Bitcoin price data. Due to the nature of the collected data, there were quite a lot of inconsistencies and null values produced. Even after cleaning, transformation, and aggregation, the quality of data still left to wish the best, but good enough for some initial analysis [1].

First attempted model was seasonal ARIMA model. As observed in Autocorrelation and Partial autocorrelation functions' plots and qqplots, the data required further transformation to become more stationary and normal. To Normalize the data I took a log() of the numeric values of the average price of Bitcoin, and to

make it more stationary I differentiated the data twice. After some diagnosis, I came to conclusion that it would be best to keep data differentiated once, so stick with using dlx. After looking up the autocorrelation and partial autocorrelation functions of dlx, and looking at diagnostic plots for it. By examining acf and pacf plots came to conclusion that there is 0 seasonality in the data, but extremely high volatility. I ended up using ARIMA(5,1,4)X(0,0,0) [2]

GARCH Model ended up being the best model to fit this dataset, despite the low quality of the initial dataset it managed to work around it perform analysis. Not sure if I agree with the results of the forecast, but MSE and MAE showed a relatively small error which might suggest the acceptable accuracy of the fit.

Overall, I'd suggest trying different coefficients for ARMA and GARCH to play with and maybe get different results to compare with mine and certainly try applying various other forecasting Time Series models to see if they can do a better job at predicting the price of Bitcoin more close to Truth.

CODE

```
#Packages
library(astsa)
library(forecast)
library(readr)
library(tidyverse)
library(tidyr)
library(tinytex)
library(fGarch)
library(timeSeries)

data <- read.csv("bitstampUSD_1-min_data_2012-01-01_to_2021-03-31.csv")

print(paste("Starting date/time is:", min(data$Timestamp)), quote=FALSE)
print(paste("Final date/time is:", max(data$Timestamp)), quote=FALSE)

n <- nrow(data)
print(paste("There are",n,"rows in our dataset"), quote= FALSE, sep = " ")

head(data)

data1 <- drop_na(data)
print(paste("After removing all the rows with missing values we now have", nrow(data1), "rows in the da
print(head(data1))

data1$Timestamp <- as.Date(as.POSIXct(data1$Timestamp, origin="1970-01-01"))
data1$Day <- format(as.Date(data1$Timestamp), "%d")
data1$Month <- format(as.Date(data1$Timestamp), "%m")
data1$Year <- format(as.Date(data1$Timestamp), "%y")
data1 <- data1[,-1]
head(data1)

data2 = data1 %>%
  group_by(Day, Month, Year) %>%
  summarise(Open = mean(Open), High = mean(High), Low = mean(Low), Close = mean(Close), VolBTC = mean(V
d2 <- arrange(data2, data2$Year, data2$Month, data2$Day, by_group = TRUE)
head(d2, n = 15)
print(paste("There are", nrow(d2), "rows in the final cleared dataframe"), quote = FALSE)
```

```
for(i in 1:length(d2$Month)){
  if(d2$Month[[i]]=="01"){
    d2$Month[[i]] <- "Jan"
  }else if(d2$Month[[i]]=="02"){
    d2$Month[[i]] <- "Feb"
  }else if(d2$Month[[i]]=="03"){
    d2$Month[[i]] <- "Mar"
  }else if(d2$Month[[i]]=="04"){
    d2$Month[[i]] <- "Apr"
  }else if(d2$Month[[i]]=="05"){
    d2$Month[[i]] <- "May"
  }else if(d2$Month[[i]]=="06"){
    d2$Month[[i]] <- "Jun"
  }else if(d2$Month[[i]]=="07"){
    d2$Month[[i]] <- "Jul"
  }else if(d2$Month[[i]]=="08"){
    d2$Month[[i]] <- "Aug"
  }else if(d2$Month[[i]]=="09"){
    d2$Month[[i]] <- "Sep"
  }else if(d2$Month[[i]]=="10"){
    d2$Month[[i]] <- "Oct"
  }else if(d2$Month[[i]]=="11"){
    d2$Month[[i]] <- "Nov"
  }else if(d2$Month[[i]]=="12"){
    d2$Month[[i]] <- "Dec"
  }
}
d2$Year <- sub("", "20", d2$Year)
d2$Year <- as.integer(d2$Year)
d2$Day <- as.integer(d2$Day)
head(d2)

d3 <- d2[,c(2,1,3,10)]
d3 <- d3[-1,]
head(d3, n = 15)

a <- list()
#d3$WeighPrice
for(i in 1:length(d3$WeighPrice)){
  a<-append(a,as.numeric(d3$WeighPrice[[i]]))
}
ts_data <- ts(a, start = c(2012,1,1), end = c(2021,3,31), frequency = 365)
head(ts_data)

plot.ts(ts_data, ylab = "Weighted Price")

qqnorm(as.numeric(ts_data))

x = as.numeric(ts_data)
lx = log(x); dlx = diff(lx); ddlx = diff(dlx, 365);
plot.ts(cbind(x, lx, dlx, ddlx))

par(mfrow=c(2,1))
```

```
monthplot(dlx, ylab="dlx")
monthplot(ddlx, ylab = "ddlx")

print(paste("The variance of dlx: ", sd(dlx)**2), quote = FALSE)
print(paste("The variance of ddlx: ", sd(ddlx)**2), quote = FALSE)

acf2(dlx, 50)

auto.arima(x)

fit <- sarima(x, 10, 1, 16) #p-values must be above the margin
fit

ts_data2 <- ts(a, start = c(2012,1,1), end = c(2020,3,31), frequency = 365)
ts3 <- as.numeric(ts_data2)
x1 = ts3
lx1 = log(x1); dlx1 = diff(lx1); ddlx1 = diff(dlx1, 365);
plot.ts(cbind(x1, lx1, dlx1, ddlx1))

sd(dlx1)**2
sd(ddlx1)**2

acf2(ddlx1)

sarima.for(ts3,n.ahead = 365, p=20, d=1, q=33)

ts_dlx <- ts(dlx, start = c(2012,1,1), end = c(2020,3,31), frequency = 365)
adf.test(ts_dlx)

acf2(residuals(fit, standardize=TRUE)**2, 50)

garch_forecast <- predict(fit, n.ahead = 30)
plot(x*exp(garch_forecast$meanForecast), main = "Bitcoin Price Forecast", xlab = "Time",
     ylab = "Avg price of Bitcoin")

mean(abs(garch_forecast$meanForecast - ts_dlx[(length(ts_dlx)-29):length(ts_dlx)]))

mean((garch_forecast$meanForecast - ts_dlx[(length(ts_dlx)-29):length(ts_dlx)])^2)
```

REFERENCES

1. https://www.kaggle.com/datasets/mczielinski/bitcoin-historical-data
2. Time Series Analysis and Its Applications with R Examples by R. H. Shumway and D. S. Stoffer 3rd Edition