Bitcoin Price Analysis and Forecasts

Kiran Benny

5/9/2019

Introduction

Bitcoin is the most popular cryptocurrency, dominating the crypto space with its blockchain technology.It works on a peer to peer network, where no intermediaries are involved. It is the digital currency, which aims to exclude intervention of any third parties, while you are transacting. It is gaining popularity and many people have started using BTCs in real time.

Bitcoin prices are surging at a greater pace.The ever-fluctuating Bitcoin price can be a cause of concern for current and potential users/investors of Bitcoin. Through this project we aim to predict the price of Bitcoin over a week using AutoRegressive Integrated Moving Average (ARIMA) Model.

Model Building

The data used is "Cryptocurrency Historical Prices by Sudalairajkumar", sourced from Kaggle consisting of 2920 rows of information pertaining to the various attributes of Bitcoin transactions collected daily from 2010 through 2018. Our dataset comprises of 24 variables out of which we had conducted exploratory analysis in phase 1 of our project using the variables btc_market_price, btc_hash_rate and btc_difficulty. For our pridictive analysis here, we have chosen the btc_market_price as our dependent variable since we are predicting the future market price of Bitcoin.

Installing and loading the required libraries

```
chooseCRANmirror(graphics=FALSE, ind=1)
knitr::opts_chunk$set(echo = TRUE)
options(warn=-1)
install.packages("ie2misc")
```

```
## Installing package into 'C:/Users/benny/OneDrive/Documents/R/win-library/3.6'
## (as 'lib' is unspecified)
```

```
## package 'ie2misc' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\benny\AppData\Local\Temp\RtmpCuN0IB\downloaded_packages
```

```
install.packages("forecast")
```

```
## Installing package into 'C:/Users/benny/OneDrive/Documents/R/win-library/3.6'
## (as 'lib' is unspecified)
```

```
## package 'forecast' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\benny\AppData\Local\Temp\RtmpCuN0IB\downloaded_packages
```

```
install.packages("ggplot2")
```

```
## Installing package into 'C:/Users/benny/OneDrive/Documents/R/win-library/3.6'
## (as 'lib' is unspecified)
```

```
## package 'ggplot2' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\benny\AppData\Local\Temp\RtmpCuN0IB\downloaded_packages
```

```
install.packages("tseries")
```

```
## Installing package into 'C:/Users/benny/OneDrive/Documents/R/win-library/3.6'
## (as 'lib' is unspecified)
```

```
## package 'tseries' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\benny\AppData\Local\Temp\RtmpCuN0IB\downloaded_packages
```

```
install.packages("tidyverse")
```

```
## Installing package into 'C:/Users/benny/OneDrive/Documents/R/win-library/3.6'
## (as 'lib' is unspecified)
```

```
## package 'tidyverse' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\benny\AppData\Local\Temp\RtmpCuN0IB\downloaded_packages
```

```
install.packages("lattice")
```

```
## Installing package into 'C:/Users/benny/OneDrive/Documents/R/win-library/3.6'
## (as 'lib' is unspecified)
```

```
## package 'lattice' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\benny\AppData\Local\Temp\RtmpCuN0IB\downloaded_packages
```

```
install.packages("dplyr")
```

```
## Installing package into 'C:/Users/benny/OneDrive/Documents/R/win-library/3.6'
## (as 'lib' is unspecified)
```

```
##
##   There is a binary version available but the source version is
##   later:
##       binary source needs_compilation
## dplyr  0.8.3  0.8.4              TRUE
##
##   Binaries will be installed
## package 'dplyr' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\benny\AppData\Local\Temp\RtmpCuN0IB\downloaded_packages
```

```
library(ie2misc)
```

```
## Loading required package: tcltk
```

```
## Loading required package: data.table
```

```
## Loading required package: tools
```

```
## Registered S3 methods overwritten by 'qdap':
##   method               from
##   t.DocumentTermMatrix tm
##   t.TermDocumentMatrix tm
```

```
library(forecast)
```

```
## Registered S3 method overwritten by 'xts':
##   method     from
##   as.zoo.xts zoo
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
## Registered S3 methods overwritten by 'forecast':
##   method             from
##   fitted.fracdiff    fracdiff
##   residuals.fracdiff fracdiff
```

```
library(ggplot2)
library(tseries)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## v purrr   0.3.3
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::between()   masks data.table::between()
## x dplyr::filter()    masks stats::filter()
## x dplyr::first()     masks data.table::first()
## x dplyr::lag()       masks stats::lag()
## x dplyr::last()      masks data.table::last()
## x purrr::transpose() masks data.table::transpose()
```

```
library(lattice)
library(dplyr)
```

Loading the data

```
my_data <- read_csv("C:\\Users\\benny\\OneDrive\\Desktop\\Misc\\RandC\\Projects\\Bitcoin Price Analysis and Forecasts\\bitco
in_dataset.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Date = col_datetime(format = "")
## )
```

```
## See spec(...) for full column specifications.
```

Due to the effect of extreme outliers in the original dataset, median values were chosen to replace the N/A values and also, since there are very few observations recorded in the dataset from the year 2010 to 2012, we have considered data from 2012 through 2018 for our predictive analysis.

```
my_data=my_data %>%mutate(btc_trade_volume=replace_na(btc_trade_volume, median(btc_trade_volume, na.rm = TRUE)))
sample_bitcoins <- subset(my_data, format(as.Date(Date),"%Y")>=2012)
head(sample_bitcoins)
```

```
## # A tibble: 6 x 24
##   Date                btc_market_price btc_total_bitco~ btc_market_cap
##   <dttm>                         <dbl>            <dbl>          <dbl>
## 1 2012-01-01 00:00:00             5.2           8007500      41639000
## 2 2012-01-02 00:00:00             5.50          8015100      44082248.
## 3 2012-01-03 00:00:00             5.29          8023200      42442728
## 4 2012-01-04 00:00:00             5.61          8030900      45023635.
## 5 2012-01-05 00:00:00             6.40          8038250      51436762.
## 6 2012-01-06 00:00:00             7.22          8046750      58097535
## # ... with 20 more variables: btc_trade_volume <dbl>,
## #   btc_blocks_size <dbl>, btc_avg_block_size <dbl>,
## #   btc_n_orphaned_blocks <dbl>, btc_n_transactions_per_block <dbl>,
## #   btc_median_confirmation_time <dbl>, btc_hash_rate <dbl>,
## #   btc_difficulty <dbl>, btc_miners_revenue <dbl>,
## #   btc_transaction_fees <dbl>, btc_cost_per_transaction_percent <dbl>,
## #   btc_cost_per_transaction <dbl>, btc_n_unique_addresses <dbl>,
## #   btc_n_transactions <dbl>, btc_n_transactions_total <dbl>,
## #   btc_n_transactions_excluding_popular <dbl>,
## #   btc_n_transactions_excluding_chains_longer_than_100 <dbl>,
## #   btc_output_volume <dbl>, btc_estimated_transaction_volume <dbl>,
## #   btc_estimated_transaction_volume_usd <dbl>
```

Time series is a sequence of data that is recorded at regular intervals of time. Observing our data we find that it is time-dependent and recorded at regular intervals of time i.e daily which makes it a time series data and suitable for time series analysis to predict future values. We will be using the AutoRegressive Integrated Moving Average (ARIMA) Model to predict the price of Bitcoin over a week.

A brief introduction to ARIMA

ARIMA stands for auto-regressive integrated moving average and is specified by three order parameters: (p, d, q).

An auto regressive (AR(p)) component refers to the use of past values in the regression equation for a series Y. The auto-regressive parameter p specifies the number of lags used in the model. A lagged version of a time series means shifting the time base back by a given number of observations. For example, AR(2) or, equivalently, ARIMA(2,0,0), is represented as:

$$Y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + e_t$$

where φ1, φ2 are parameters for the model.

The d represents the degree of differencing in the integrated (I(d)) component. Differencing a series involves simply subtracting its current and previous values d times. Often, differencing is used to stabilize the series when the stationarity assumption is not met, which we will discuss below.

A moving average (MA(q)) component represents the error of the model as a combination of previous error terms et. The order q determines the number of terms to include in the model.

$$Y_t = c + \theta_1 e_{t-1} + \theta_2 e_{t-2} + \ldots + \theta_q e_{t-q} + e_t$$

Differencing, autoregressive, and moving average components make up a non-seasonal ARIMA model which can be written as a linear equation:

$$Y_t = c + \phi_1 y_{d\,t-1} + \phi_p y_{d\,t-p} + \ldots + \theta_1 e_{t-1} + \theta_q e_{t-q} + e_t$$

where yd is Y differenced d times and c is a constant.

Decomposing the data

Many time series include trend, cycle and seasonality.These components capture the historical patterns in the series. Not every series will have all three (or any) of these components, but if they are present, deconstructing the series can helps us understand its behavior and prepare a foundation for building our forecasting model.

Trend *A trend exists when there is a long-term increase or decrease in the data. It does not have to be linear. Sometimes we will refer to a trend as "changing direction", when it might go from an increasing trend to a decreasing trend.*

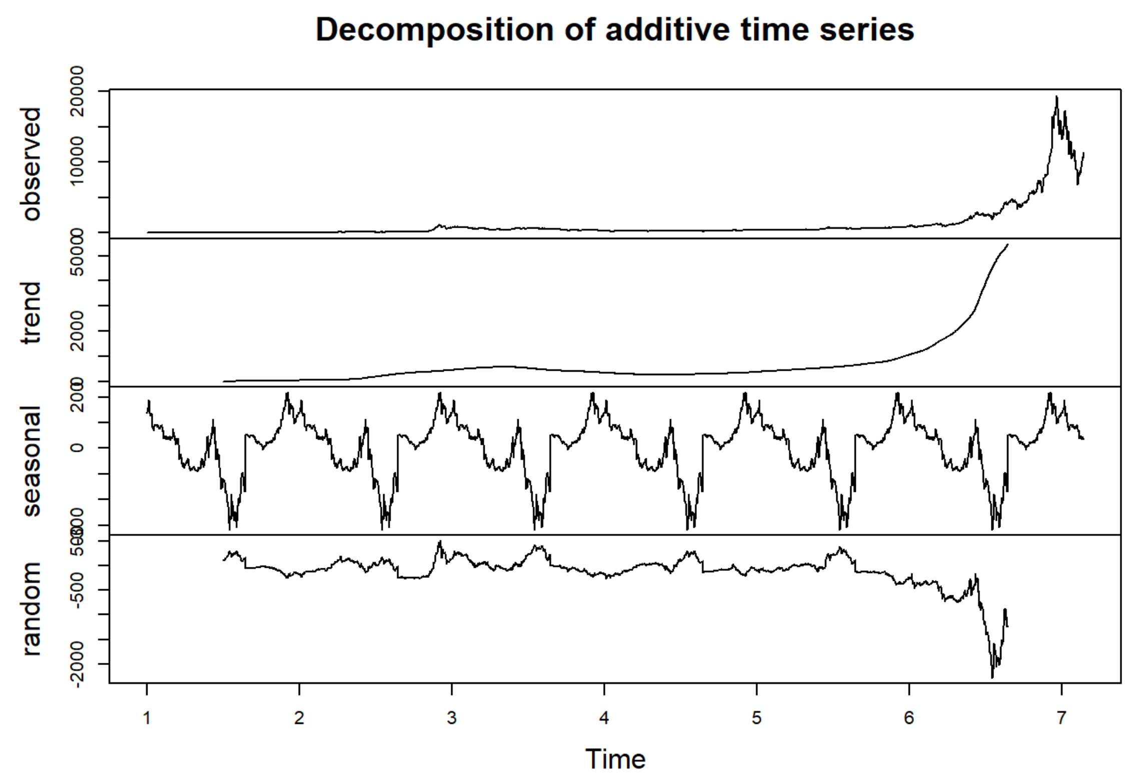
Seasonality *A seasonal pattern occurs when a time series is affected by seasonal factors such as the time of the year or the day of the week. Seasonality is always of a fixed and known frequency.*

Cyclicity *A cycle occurs when the data exhibit rises and falls that are not of a fixed frequency. These fluctuations are usually due to economic conditions, and are often related to the "business cycle".*

Finally, part of the series that can't be attributed to seasonal, cycle, or trend components is referred to as residual or error.

The process of extracting these components is referred to as decomposition.

```
tsdata<-ts(sample_bitcoins$btc_market_price, frequency = 365)
decomposed<-decompose(tsdata)
plot(decomposed)
```



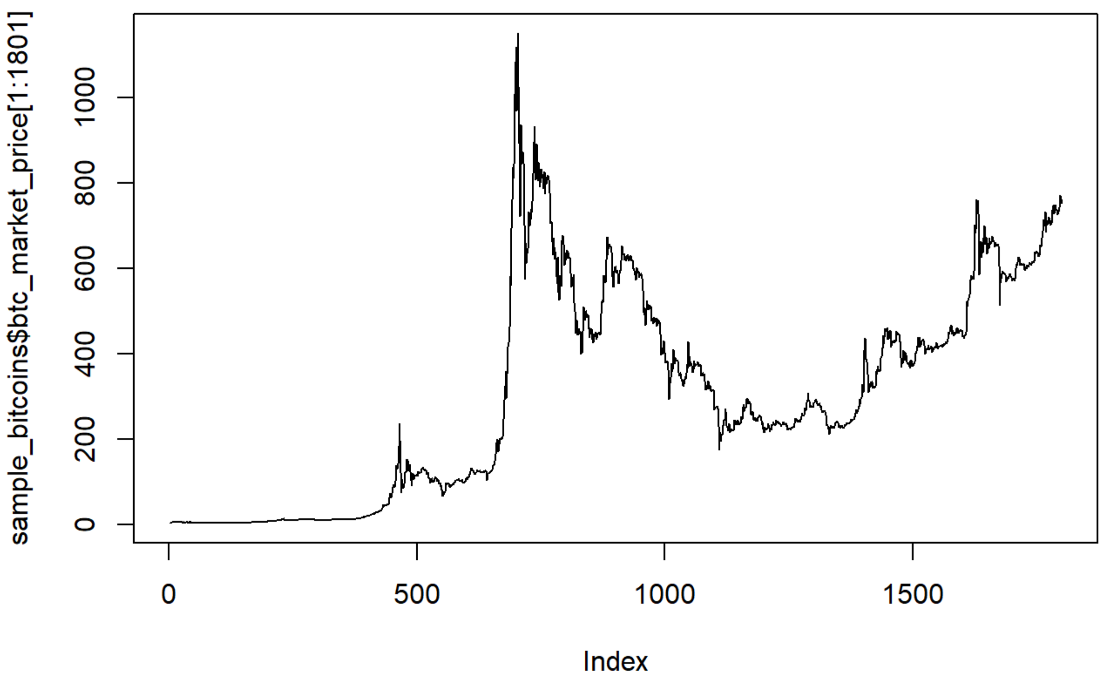
We have used the ts() function to create a time series object to pass to the decompose() function in order to analyse the series.As for the frequency parameter in ts() object, we are specifying periodicity of the data, i.e., number of observations per period. Since we are using daily data of 6 years, we have 365 observations per year.

From the graph above, we notice that the time series exhibits an increasing/positive trend as well as a seasonality component indicating that our series is non-stationary.

Stationarity

Fitting an ARIMA model requires the series to be stationary. A series is said to be stationary when its mean and variance are constant over time and there is no trend or seasonality component present. It is easier to predict when the series is stationary. Clearly, our data is non-stationary due to the presence of the trend and seasonality components.

```
plot(sample_bitcoins$btc_market_price[1:1801], type = 'l')
```



The augmented Dickey-Fuller (ADF) test is a formal statistical test for stationarity. The null hypothesis assumes that the series is non-stationary. ADF procedure tests whether the change in Y can be explained by lagged value and a linear trend. If contribution of the lagged value to the change in Y is non-significant and there is a presence of a trend component, the series is non-stationary and null hypothesis will not be rejected.

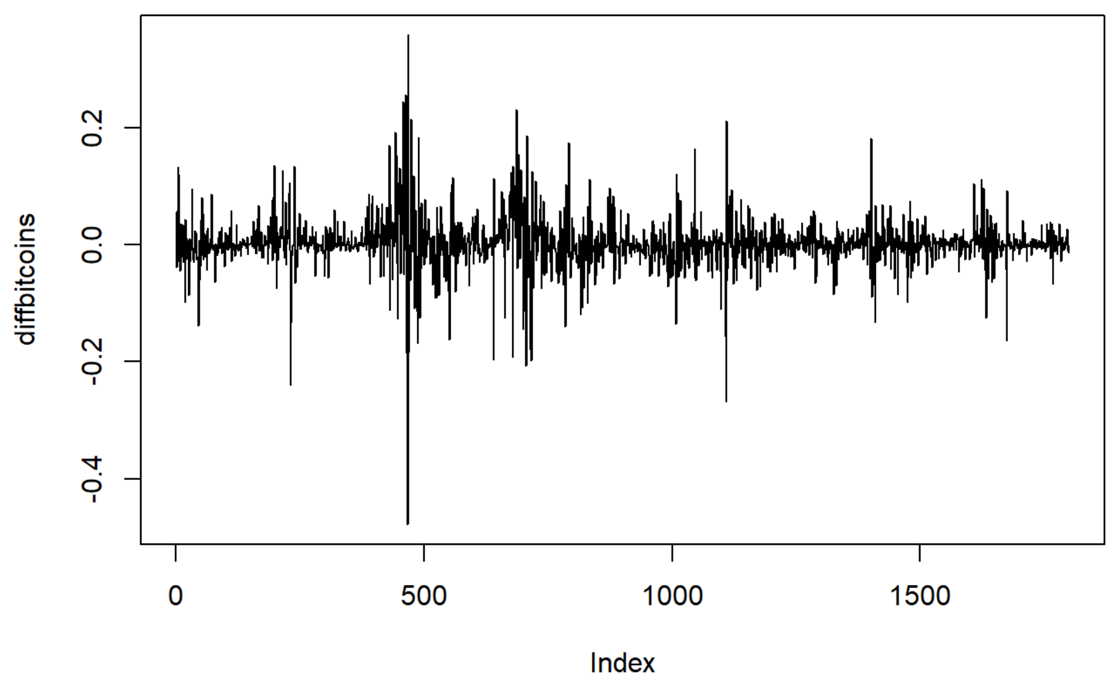
Our bitcoin data is non-stationary; the bitcoin market price changes through time, therefore, the ADF test does not reject the null hypothesis of non-stationarity, confirming that our data is non-stationary:

```
adf.test(sample_bitcoins$btc_market_price)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  sample_bitcoins$btc_market_price
## Dickey-Fuller = -0.87884, Lag order = 13, p-value = 0.9546
## alternative hypothesis: stationary
```

A non-stationary series can be stationarized by performing log transformation and differencing on the series. Transformations such as logarithms can help to stabilise the variance of a time series. Differencing can help stabilise the mean of a time series by removing changes in the level of a time series, and therefore eliminating (or reducing) trend and seasonality. The difference is calculated by subtracting one period’s values from the previous period’s values.

```
bitcoins<-log(sample_bitcoins$btc_market_price[1:1801])
diffbitcoins <- diff(bitcoins,1)
plot(diffbitcoins, type = 'l')
```



Running the augmented Dickey-Fuller (ADF) test on differenced data rejects the null hypotheses of non-stationarity. Plotting the differenced series, we see no visible strong trend. This suggests that differencing of order 1 terms is sufficient and should be included in the model.

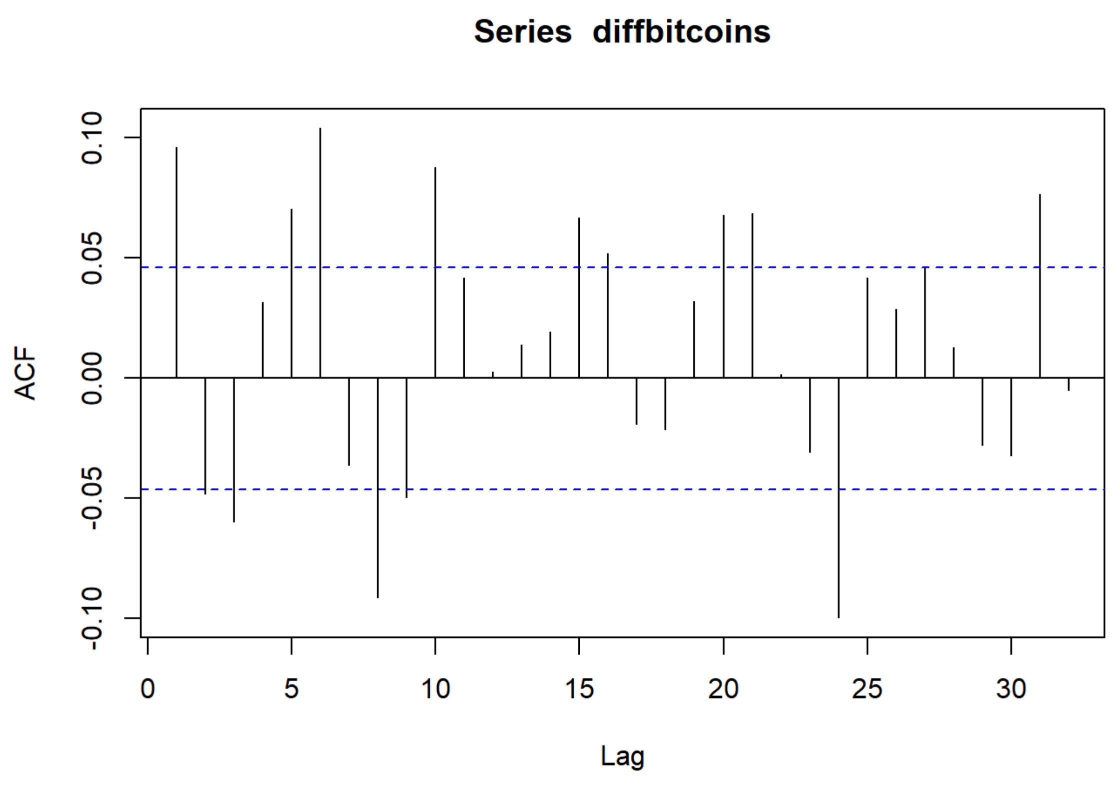
```
adf.test(diffbitcoins)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  diffbitcoins
## Dickey-Fuller = -10.77, Lag order = 12, p-value = 0.01
## alternative hypothesis: stationary
```

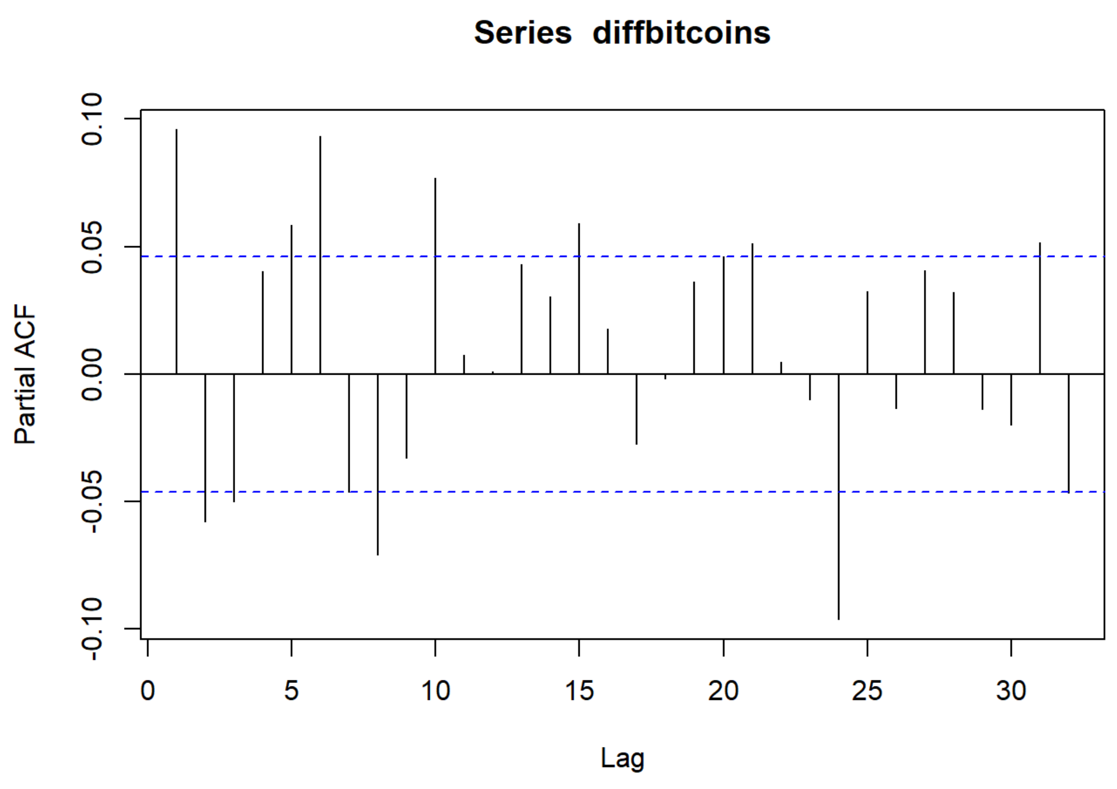
Autocorrelations

Autocorrelation plots (also known as ACF or the auto correlation function) are useful in determining whether a series is stationary.ACF plots display correlation between a series and its lags. In addition to suggesting the order of differencing, ACF plots can help in determining the order of the MA (q) model. Partial autocorrelation plots (PACF) display correlation of the residuals (which remains after removing the effects which are already explained by the earlier lag(S)) with the next lag value. PACF plots are useful when determining the order of the AR(p) model.

```
Acf(diffbitcoins)
```



```
Pacf(diffbitcoins)
```



Spikes at particular lags of the differenced series can help us choose the values of p or q for our model.

Fitting the ARIMA model

The forecast package allows the user to explicitly specify the order of the model using the arima() function, or automatically generate a set of optimal (p, d, q) using auto.arima() function. This function searches through combinations of order parameters and picks the set that will produce the best fit for the model.

There exists a number of such criteria for comparing quality of fit across multiple models. Two of the most widely used are Akaike information criteria (AIC) and Baysian information criteria (BIC). These criteria are closely related and can be interpreted as an estimate of how much information would be lost if a given model is chosen. When comparing models, minimizing the AIC and BIC is favorable.

```
tsdata1<-ts(bitcoins, frequency = 365)
price<- auto.arima(tsdata1, trace = TRUE)
```

```
##
##  Fitting models using approximations to speed things up...
##
##  ARIMA(2,1,2)(1,0,1)[365] with drift         : Inf
##  ARIMA(0,1,0)             with drift         : -6012.18
##  ARIMA(1,1,0)(1,0,0)[365] with drift         : Inf
##  ARIMA(0,1,1)(0,0,1)[365] with drift         : Inf
##  ARIMA(0,1,0)                                : -6007.524
##  ARIMA(0,1,0)(1,0,0)[365] with drift         : Inf
##  ARIMA(0,1,0)(0,0,1)[365] with drift         : Inf
##  ARIMA(0,1,0)(1,0,1)[365] with drift         : Inf
##  ARIMA(1,1,0)             with drift         : -6027.196
##  ARIMA(1,1,0)(0,0,1)[365] with drift         : Inf
##  ARIMA(1,1,0)(1,0,1)[365] with drift         : Inf
##  ARIMA(2,1,0)             with drift         : -6031.304
##  ARIMA(2,1,0)(1,0,0)[365] with drift         : Inf
##  ARIMA(2,1,0)(0,0,1)[365] with drift         : Inf
##  ARIMA(2,1,0)(1,0,1)[365] with drift         : Inf
##  ARIMA(3,1,0)             with drift         : -6034.689
##  ARIMA(3,1,0)(1,0,0)[365] with drift         : Inf
##  ARIMA(3,1,0)(0,0,1)[365] with drift         : Inf
##  ARIMA(3,1,0)(1,0,1)[365] with drift         : Inf
##  ARIMA(4,1,0)             with drift         : -6042.301
##  ARIMA(4,1,0)(1,0,0)[365] with drift         : Inf
##  ARIMA(4,1,0)(0,0,1)[365] with drift         : Inf
##  ARIMA(4,1,0)(1,0,1)[365] with drift         : Inf
##  ARIMA(5,1,0)             with drift         : -6050.914
##  ARIMA(5,1,0)(1,0,0)[365] with drift         : Inf
##  ARIMA(5,1,0)(0,0,1)[365] with drift         : Inf
##  ARIMA(5,1,0)(1,0,1)[365] with drift         : Inf
##  ARIMA(5,1,1)             with drift         : -6052.028
##  ARIMA(5,1,1)(1,0,0)[365] with drift         : Inf
##  ARIMA(5,1,1)(0,0,1)[365] with drift         : Inf
##  ARIMA(5,1,1)(1,0,1)[365] with drift         : Inf
##  ARIMA(4,1,1)             with drift         : -6049.848
##  ARIMA(5,1,2)             with drift         : -6065.295
##  ARIMA(5,1,2)(1,0,0)[365] with drift         : Inf
##  ARIMA(5,1,2)(0,0,1)[365] with drift         : Inf
##  ARIMA(5,1,2)(1,0,1)[365] with drift         : Inf
##  ARIMA(4,1,2)             with drift         : -6063.137
##  ARIMA(5,1,3)             with drift         : -6059.232
##  ARIMA(4,1,3)             with drift         : -6061.137
##  ARIMA(5,1,2)                                : -6062.112
##
##  Now re-fitting the best model(s) without approximations...
##
##  ARIMA(5,1,2)             with drift         : -6070.031
##
##  Best model: ARIMA(5,1,2)             with drift
```

```
price
```

```
## Series: tsdata1
## ARIMA(5,1,2) with drift
##
## Coefficients:
##          ar1      ar2     ar3      ar4     ar5      ma1     ma2   drift
##       0.6618  -0.8749  0.0598  -0.0139  0.0459  -0.5698  0.7882  0.0028
## s.e.  0.0685   0.0668  0.0356   0.0322  0.0275   0.0649  0.0591  0.0011
##
## sigma^2 estimated as 0.001998:  log likelihood=3044.07
## AIC=-6070.13   AICc=-6070.03   BIC=-6020.67
```

We can see that the auto.arima() function has chosen ARIMA(5,1,2) as the best model for our analysis.

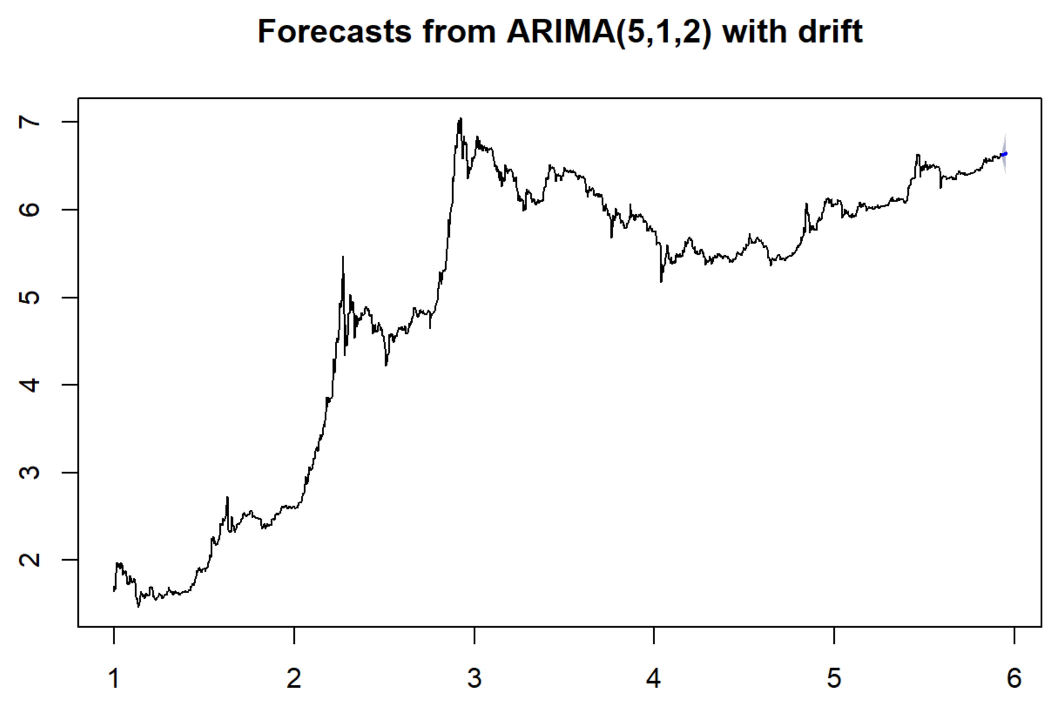
Forecasting

Forecast() is a generic function for forecasting from time series or time series models. We can specify forecast horizon h periods ahead for predictions to be made, and use the fitted model to generate those predictions. We have set the h value as 7 as we will be predicting the price of bitcoin a week ahead.

```
forecastedvalues <- forecast(price, h=7)
forecastedvalues
```

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 5.934247       6.629207 6.571929 6.686484 6.541608 6.716805
## 5.936986       6.634889 6.550082 6.719696 6.505188 6.764590
## 5.939726       6.637882 6.533381 6.742382 6.478062 6.797701
## 5.942466       6.638403 6.518448 6.758359 6.454948 6.821859
## 5.945205       6.638935 6.505570 6.772300 6.434971 6.842900
## 5.947945       6.642187 6.495016 6.789358 6.417108 6.867266
## 5.950685       6.647245 6.486149 6.808341 6.400870 6.893620
```

```
plot(forecastedvalues)
```



```
summary(forecastedvalues)
```

```
##
## Forecast method: ARIMA(5,1,2) with drift
##
## Model Information:
## Series: tsdata1
## ARIMA(5,1,2) with drift
##
## Coefficients:
##          ar1      ar2     ar3      ar4     ar5      ma1     ma2   drift
##       0.6618  -0.8749  0.0598  -0.0139  0.0459  -0.5698  0.7882  0.0028
## s.e.  0.0685   0.0668  0.0356   0.0322  0.0275   0.0649  0.0591  0.0011
##
## sigma^2 estimated as 0.001998:  log likelihood=3044.07
## AIC=-6070.13   AICc=-6070.03   BIC=-6020.67
##
## Error measures:
##                        ME       RMSE        MAE         MPE      MAPE
## Training set -1.52339e-05 0.04458228 0.02593322 0.002468338 0.6092398
##                    MASE         ACF1
## Training set 0.01781091 -0.002149899
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 5.934247       6.629207 6.571929 6.686484 6.541608 6.716805
## 5.936986       6.634889 6.550082 6.719696 6.505188 6.764590
## 5.939726       6.637882 6.533381 6.742382 6.478062 6.797701
## 5.942466       6.638403 6.518448 6.758359 6.454948 6.821859
## 5.945205       6.638935 6.505570 6.772300 6.434971 6.842900
## 5.947945       6.642187 6.495016 6.789358 6.417108 6.867266
## 5.950685       6.647245 6.486149 6.808341 6.400870 6.893620
```

Trasforming the data back to its original “scale” for comparison of the predicted prices with the actual prices through the exp() function that computes antilogarithm:

```
forecastedvaluesextracted <- as.numeric(forecastedvalues$mean)
finalforecastedvalues <- exp(forecastedvaluesextracted)
```

Model Results

The table below shows a comparison of our predicted prices vs the actual price of bitcoin. We can see that there is a very nominal difference between the Actual Price of Bitcoin vs The Forecasted Price obtained through our ARIMA model which means our model is a good fit.

```
df <- data.frame(sample_bitcoins$btc_market_price[1801:1807],finalforecastedvalues)
col_headings <- c("Actual Price","Forecasted Price")
names(df) <- col_headings
attach(df)
df
```

```
##   Actual Price Forecasted Price
## 1     754.6398         756.8815
## 2     756.6215         761.1946
## 3     766.1166         763.4760
## 4     769.7297         763.8744
## 5     770.0281         764.2807
## 6     773.4013         766.7702
## 7     768.3031         770.6580
```

Accuracy

The mean absolute percentage error (MAPE), also known as mean absolute percentage deviation (MAPD), is a measure of prediction accuracy of a forecasting method.It measures this accuracy as a percentage, and can be calculated as the average absolute percent error for each time period minus actual values divided by actual values:

Calculating the MAPE manually:

```
percentage_error <- ((abs(df$'Actual Price'-df$'Forecasted Price')/(df$'Actual Price'))*100)
percentage_error
```

```
## [1] 0.2970570 0.6044082 0.3446785 0.7607081 0.7463959 0.8573842 0.3065086
```

```
mean(percentage_error)
```

```
## [1] 0.5595915
```

R provides a function that can calculate the MAPE directly through the mape() function:

```
error<-mape(`Forecasted Price`, `Actual Price`)
error
```

```
## [1] 0.5595915
```

From the results obtained above, we observe an error percentage of 0.55 which is fairly good considering our model’s fit.

Another way to validate our model is by using The Ljung-Box Test. The Ljung-Box test tests the “overall” randomness based on a number of lags. For this reason, it is often referred to as a “portmanteau” test. We apply the Box-Ljung test to the residuals from the ARIMA(5,1,2) model fit to determine whether residuals are random.

```
Box.test(price$residuals,lag=5,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  price$residuals
## X-squared = 4.2364, df = 5, p-value = 0.5159
```

```
Box.test(price$residuals,lag=10,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  price$residuals
## X-squared = 13.636, df = 10, p-value = 0.1902
```

```
Box.test(price$residuals,lag=25,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  price$residuals
## X-squared = 56.232, df = 25, p-value = 0.0003384
```

The Box-Ljung test for different lags shows that the first 5 lag autocorrelations among the residuals are greater than 0.05 (p-value = 0.5159), indicating that the residuals are random and that this model is the best fit.

Conclusion

In this report, we have tried to predict the price of Bitcoin through ARIMA modelling. We have observed that the ARIMA(5,1,2) is the best fit model for our predictive analysis. We find that there is a very nominal difference between the Actual Price of Bitcoin vs The Forecasted Price obtained through our ARIMA model with a mean percentage error of 0.55 which means our model is a good fit and our prediction has been successful.

Other forecasting techniques, such as exponential smoothing, could probably help make the model more accurate using a weighted combinations of seasonality, trend, and historical values to make predictions. In addition, daily bitcoin price data is probably highly dependent on other factors, such as market capital, hash rate, difficulty etc. We could try fitting time series models that allow for inclusion of other predictors using methods such ARMAX or dynamic regression in order to get an even better forecast.