In this study, we explain how to fit an ARIMA model to our data using R. ARIMA models have a reputation for being transformative and employing historical data for predictions. Here, we want to fit a model to our Bitcoin data in order to be able to do forecasting.

Introduction to ARIMA

ARIMA is an abbreviation of auto-regressive integrated moving average and its main components are (P, d, q) parameters.

The**auto regressive (AR(p))** component is referring to the use of past values in the regression equation for the output series. The auto-regressive parameter p specifies the number of lags used in 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.

A **moving average (MA(q))** component represents the error of the model as a combination of previous error terms et.

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

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

ARIMA model limitations

These models are suitable for long and robust series where the present values are dependent on the past values.

Fitting a model using ARIMA

In order to fit a model to our data using ARIMA, the following steps should be followed:

1. Data Examination

* Plotting the data in order to find patterns and irregularities
* Dropping the missing values and outliers (tsclean() is a suitable method for this purpose)
* Calculating the logarithm of the time series to see a growth trend

1. Data Analyzing

* Finding trends or seasonality in the data if there is any
* Applying decompose() or stl() for removing the components of the series

1. Stationarity

* Is there any stationarity in the data?
* Use **adf.test()**, ACF, PACF plots to determine order of differencing needed

1. Autocorrelations and selecting model order

* Finding out the order of ARIMA using ACF() and PACF() plots

1. Fit an ARIMA model
2. Assess and repeat

* Examine the residuals to make sure there is no pattern and they are normally distributed
* In case of existing any visible patterns, draw ACF/ PACF plots to figure out if any additional parameters is needed.
* Refit model if it is necessary. Testing model errors and fit criteria such as AIC or BIC.
* Compute fitting model by means of the selected model.

Step 1: Load R packages

In this study for fitting ARIMA model, we will be using the following libraries. Moreover, we will be utilizing “btceUSD\_1-min\_data\_2012-01-01\_to\_2017-05-31.csv” file as our dataset.

setwd**(**'F:/Courses/Modeling\_Simulation/'**)**

install.packages**(**'tseries'**)**

install.packages**(**'forecast'**)**

library**(**'ggplot2'**)**

library**(**'forecast'**)**

library**(**'tseries'**)**

btc**<-** read.csv**(**"bitcoin.csv",header**=TRUE**, stringsAsFactors**=FALSE)**

Step 2: Data Cleaning

Since our dataset contains too many Nan values, we will be using a method to replace them with their closest values. Then, we changed the intervals of the Bitcoin opening values from minute to day to make our data more visually appealing. Furthermore, we transformed the timestamp to date.

#Replacing the Nan opening values with the nearest values

btc**$**openC**<-**replaceNanNearst**(**btc**$**Open**)**

#Changing the interval from minute to minute to day to day

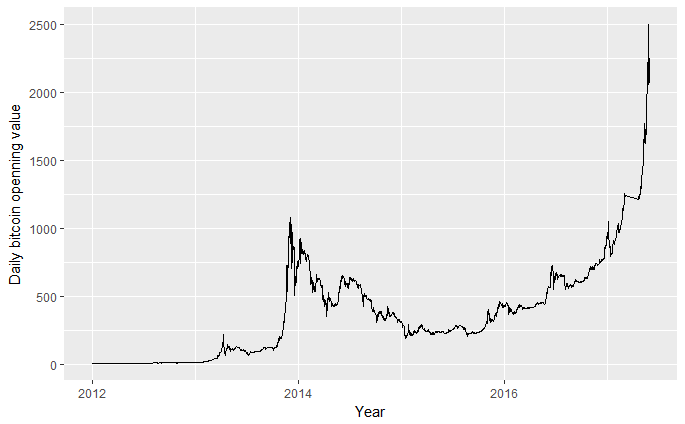
sq**<-**seq**(**1,length**(**btc**$**openC**)**,by **=** 1440**)**

btc\_date**<-**btc**$**Timestamp**[**sq**]**

BTC**<-** data.frame**(**btc\_open**<-**btc**$**openC**[**sq**]**,btc\_Date**<-**as.Date**(**as.POSIXct**(**btc\_date, origin **=** "1970-01-01", tz **=** "GMT"**)))**

Step 3: Analyzing the data

Plotting the data to find the outliers and irregularities is of importance.



In some cases, the opening value of Bitcoin increased to 1125 USD on month and dropped to 500 USD the next month. These are doubted outliers that might incline the model by deviating the summaries. Tsclean() in R provides facilities for passing up these outliers. This method replaces outliers applying series smoothing and decomposition. For this purpose, first a time series object should be created using ts() command to be passed to tsclean():

#Removing the outliers

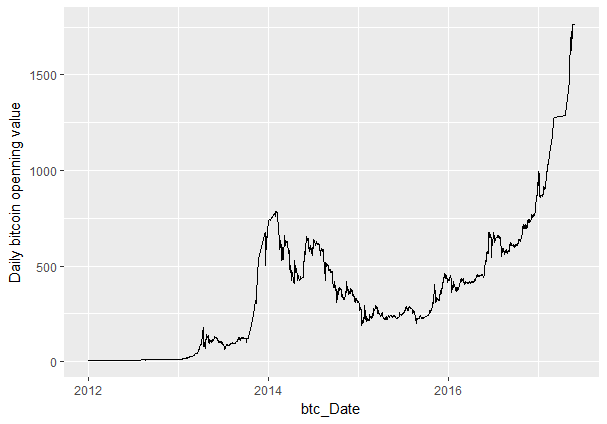
count\_ts **=** ts**(**BTC**[**, c**(**'btc\_open'**)])**

BTC**$**clean\_open **=** tsclean**(**count\_ts**)**

ggplot**()** **+**

geom\_line**(**data **=** BTC, aes**(**x **=** btc\_Date, y **=** clean\_open**))** **+** ylab**(**"Daily bitcoin openning value"**)**

After plotting the clean series using ggplot:



It is noticeable that even after ignoring the outliers, there are still fluctuations in the Bitcoin opening values. One of the promising solutions to this issue is through considering a line connecting its bigger bottoms and peaks when smoothing the fluctuations. This concept is known as moving average in time series. This meaning calculates the average of the points across different intervals and therefore makes the data a more predictable series.

A moving average (MA) of order m can be calculated using the following formula:

Where y is the series, k is the number of periods around each point and m=2k+1.

It is worth noting that the moving average here is completely different from the MA(q) component in the definition of ARIMA. MA(q) is related to error lags but the summary statistic of moving average is a part of the data smoothing technique.

We have a smoother function as we expand the size of the window for the moving average. If we take monthly instead of weekly moving average, we can have a more predictable series.

#Making the clean data(The data with no outliers) smoother using the moving average

BTC**$**open\_ma **=** ma**(**BTC**$**clean\_open, order**=**7**)** # using the clean count with no outliers

BTC**$**ooen\_ma30 **=** ma**(**BTC**$**clean\_open, order**=**30**)**

BTC**$**open\_ma120 **=** ma**(**BTC**$**clean\_open, order**=**120**)**

ggplot**()** **+**

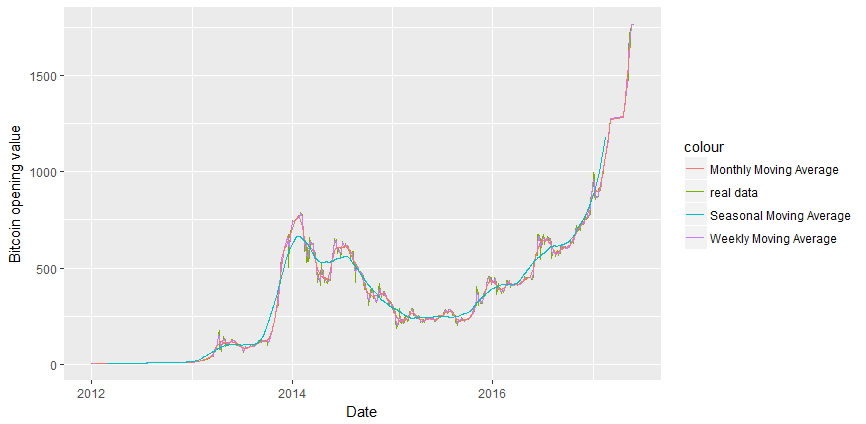
geom\_line**(**data **=** BTC, aes**(**x **=** btc\_Date, y **=** clean\_open, colour **=** "real data"**))** **+**

geom\_line**(**data **=** BTC, aes**(**x **=** btc\_Date, y **=** cnt\_ma, colour **=** "Weekly Moving Average"**))** **+**

geom\_line**(**data **=** BTC, aes**(**x **=** btc\_Date, y **=** cnt\_ma30, colour **=** "Monthly Moving Average"**))** **+**

geom\_line**(**data **=** BTC, aes**(**x **=** btc\_Date, y **=** cnt\_ma120, colour **=** "Seasonal Moving Average"**))** **+**xlab**(**"Date"**)+**

ylab**(**"Bitcoin opening value"**)**



As can be seen, the Seasonal Moving Average offers the smoothest data and we will model it (as shown by green line above) for simplicity.

Step 4: Decompose the data

The corner stones for analyzing a time series are seasonality, trend and cycle. These components detect the historical patterns of the series. In case of being available, these components can contribute in analyzing the behavior of the series in order to be able to create a forecasting model.

**Seasonal component** is related to the fluctuations in the data in seasonal cycles. For example, in which seasons, the Bitcoin value is larger and in which seasons in has smaller values.

**Trend component** is assigned to the whole pattern of the series. Does the series have an increasing or decreasing trend over all of the intervals?

**Cycle component** is about all of the increasing or decreasing trends that are not seasonal. In general, trend and cycle components are considered together as one.

The last part of the series that cannot be referred to any of the other components is called residual or error.

The process of taking out these components is called **decomposition**.

For example, if Y is the Bitcoin opening value, we can decompose the series by applying an additive or multiplicative model:

Where S, T and E are seasonal, trend and error components.

An additive model is more appropriate for the conditions that seasonal or trend components are not proportional to the level of series. In these situations, we can just overlay the components together to rebuild the series. In contrast, if the seasonality is dependent to trend of the series, a multiplicative model can be more efficient.

As fitting a model to seasonal data is a complicated process, we will explain how to de-seasonalize the series and use a “vanilla” non-seasonal ARIMA model.

In order to do this, we first calculate seasonal component of the data using stl(). Stl() is a function for decomposing and forecasting the series.  It calculates the seasonal component of the series using smoothing, and configuring the series by subtracting seasonality.

#Decomposion

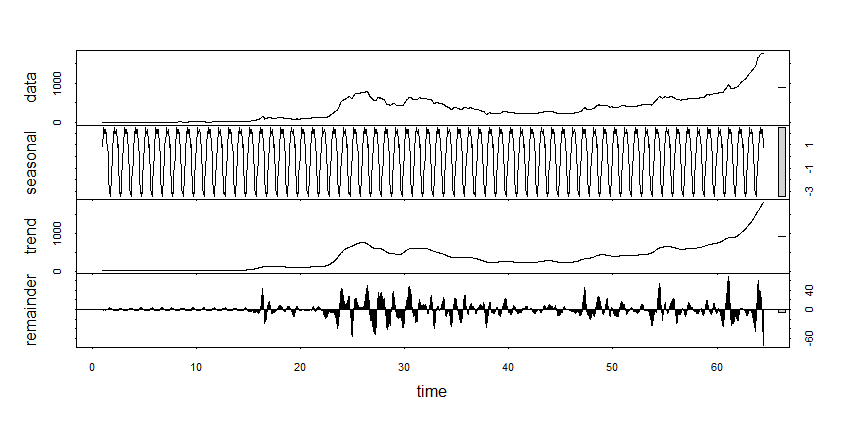
count\_ma **=** ts**(**na.omit**(**BTC**$**open\_ma**)**, frequency**=**30**)**

decomp **=** stl**(**count\_ma, s.window**=**"periodic"**)**

deseasonal\_open **<-** seasadj**(**decomp**)**

plot(decomp)

Seasadj() is a method for removing the seasonality. We set the periodicity of the data (i.e., the number of observations per period) for the frequency parameter in ts() object. As we are utilizing smoothed daily data, we have 30 observations per month.



Step 5: Stationarity

Fitting an ARIMA model requires series to be stationary which means that its mean, variance and autocovariance are independent from time. One of the formal statistical tests for stationarity is the augmented Dickey-Fuller (ADF). The null hypothesis here is that the series is non-stationary. ADF checks if the change in the Y values can be described by lagged value and a linear trend. If values of Y are related to lagged values and a trend component emerges, the series is considered to be non-stationary and null hypothesis will not be rejected.

#ADF Test for testing stationarity

**>** adf.test**(**count\_ma, alternative **=** "stationary"**)**

Augmented Dickey**-**Fuller Test

data**:** count\_ma

Dickey**-**Fuller **=** 0.85419, Lag order **=** 12, p**-**value **=** 0.99

alternative hypothesis**:** stationary

The opening values of Bitcoin are non-stationary. The average of opening values of Bitcoin change through time. The ADF test does not reject the NULL hypothesis and corroborates our guess.

The non-stationary series can be transformed to stationary ones by deducting one period's values from the previous period's values. Differencing can assist in deleting the trends or cycles from the data and makes it stationary:

In a similar vein, differencing can be used for removing seasonal patterns at specific lags:

d component of ARIMA represents the number of carried out differences.

Step 6: Autocorrelations and Choosing Model Order

In this section, the methods for choosing the order of differencing will be explained. Auto Correlation Function (ACF) is an efficient method for determining if a series is stationary. It is also beneficial in choosing the order of the parameters of the ARIMA model. In case of existing a correlation between a variable and its lags, there are some trends and therefore, the series changes over time.

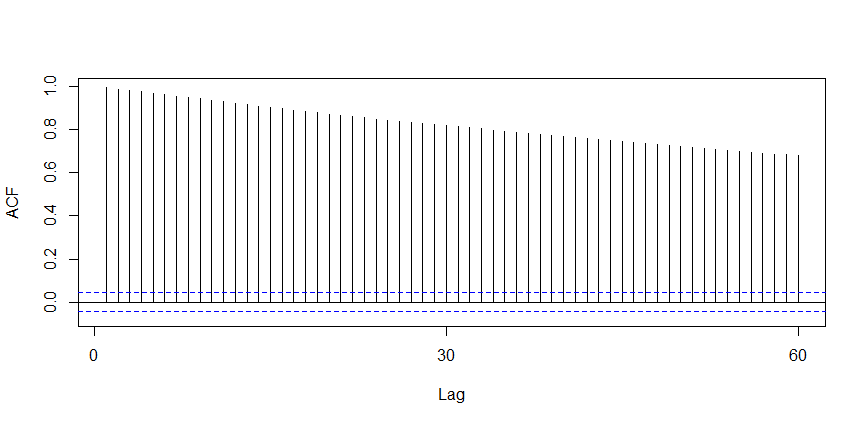
ACF plots show the correlation between a series and its lags. Partial autocorrelation plots (PACF) show the correlation between a variable and its lags that is not included by previous lags.

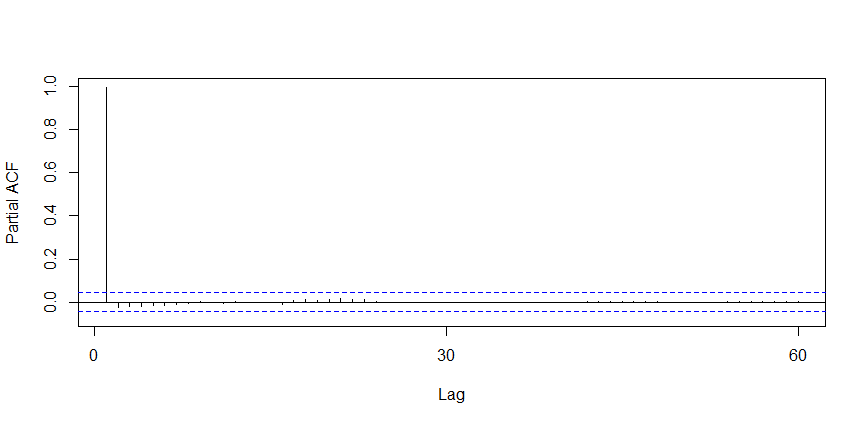
As can be seen, there are autocorrelations with many lags in Bitcoin opening values by the ACF plot. This is because of the carry-over correlation from the early lags. However, the PACF plot does not show any spikes.

#Plotting ACF and PACF plots for displaying the auto correlation

Acf**(**count\_ma, main**=**''**)**

Pacf**(**count\_ma, main**=**''**)**





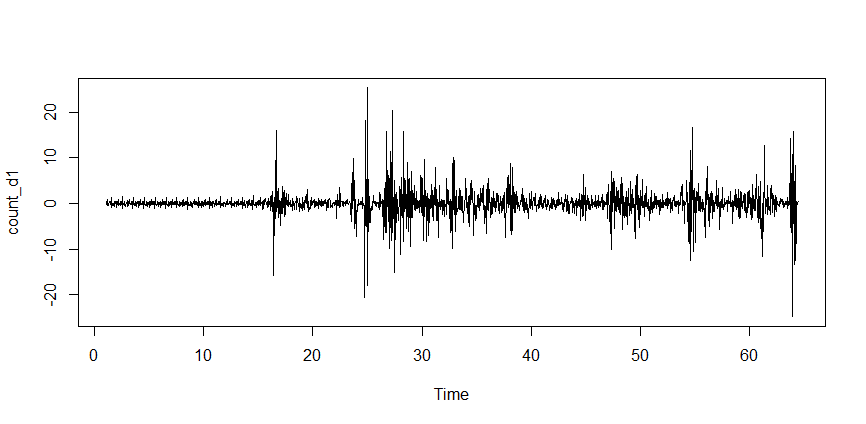
We will start from d=1 and reevaluate if more differencing is needed. The null hypothesis of being non-stationary is rejected with the adf test. Considering the plotted differenced series, there is an oscillating pattern around 0 with no visible strong trend. Therefore, differencing of order 1 terms seems to be sufficient but differencing of order 2 decrease the number spikes in ACF so it should be included in the model.

#Plotting the differenced series for choosing the order of differencing

count\_d1 **=** diff**(**deseasonal\_open, differences **=** 2**)**

plot**(**count\_d1**)**

adf.test**(**count\_d1, alternative **=** "stationary"**)**

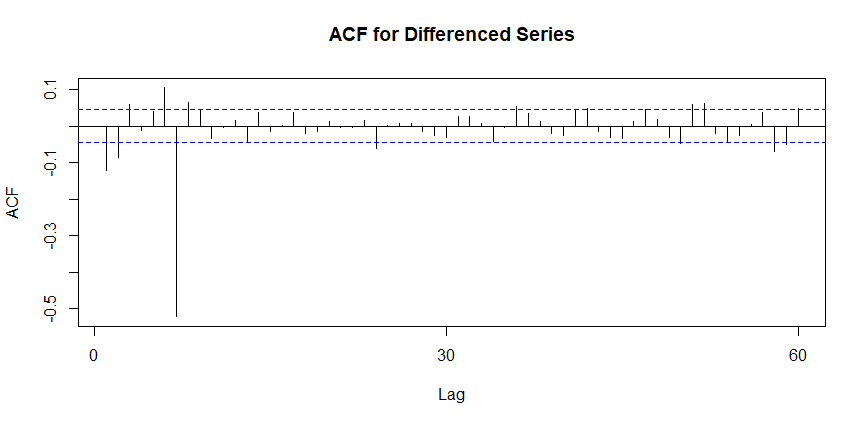


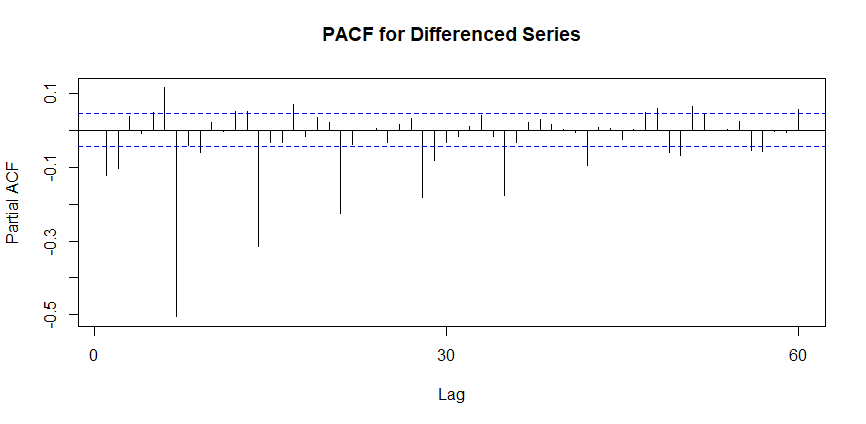
Next, spikes at particular lags of the differenced series could be of benefit when it comes to choosing the p or q for our model:

#Plotting ACF and PACF plots of the differencing series to choose p and q for the model

Acf**(**count\_d1, main**=**'ACF for Differenced Series'**)**

Pacf**(**count\_d1, main**=**'PACF for Differenced Series'**)**





There are noticeable auto correlations at lag1, 2 and 7. In PACF plot, there are significant spikes at lag 1, 2 and 7 and 14. Therefore, we would test models with AR or MA components of order 1, 2, 7 or 14. A spike at lag 7 and 14 may imply that there is a seasonal pattern available, perhaps as day of the week.

Step 7: Fitting an ARIMA model

We should use the **forecast** package to apply the arima() function for determining order of the model. We can also automatically specify a set of optimal (p, d, q) using auto.arima().This function searches through combinations of order parameters and picks the set that optimizes model fit criteria.

There are some criteria for evaluating the quality of fitting models. Among the most reputable ones, we can mention Akaike information criteria (AIC) and Baysian information criteria (BIC). These criteria can examine the amount of information lost for the chosen model. The objective here is that the selected minimizes AIC and BIC.

For applying auto.arima(), it is necessary to complete steps 1-5 to comprehend the series and analyzing model results. We can also determine maximum order of (p, d, q) using auto.arima() that is by default equal to 5.

#Fitting the model

**>** auto.arima**(**deseasonal\_open, seasonal**=FALSE)**

Series**:** deseasonal\_open

ARIMA**(**2,2,2**)**

Coefficients**:**

ar1 ar2 ma1 ma2

**-**1.1647 **-**0.8839 1.0949 0.7582

s.e. 0.0392 0.0237 0.0564 0.0308

sigma**^**2 estimated as 7.191**:** log likelihood**=-**4575.56

AIC**=**9161.11 AICc**=**9161.14 BIC**=**9188.87

The offered parameters (2, 2, 2) by the automated procedure are compatible with our expectations from the explained steps. The model incorporates differencing of degree 2, and applies an autoregressive term of the second lag and a moving average model of order 2.

Our model regarding the ARIMA notation can be defined as follows:

Here, E is some error and the original series is differenced with order 2.

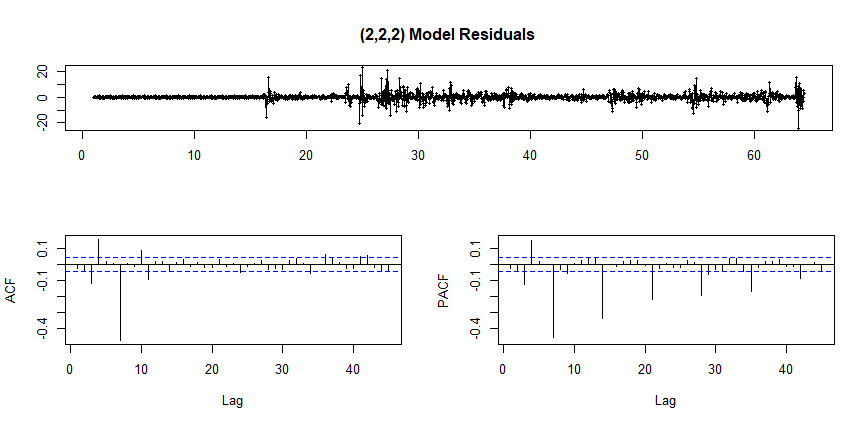
Step 8: Evaluate and Iterate

So far, we fitted a model to our data that can be used for forecasting, but how much we can trust it? For this purpose, we can test ACF and PACF plots for model residuals. In case of correctly selecting model order parameters and structure, there should not be any significant autocorrelations.

#Testing the model using model residuals

fit**<-**auto.arima**(**deseasonal\_open, seasonal**=FALSE)**

tsdisplay**(**residuals**(**fit**)**, lag.max**=**45, main**=**'(2,2,2) Model Residuals'



From the ACF/PACF and model residuals plots , we can see that there is a repeating pattern repeating at lag 7. This suggests that our model might work better with a different configuration, such as p = 7 or q = 7.

We can do the fitting process from the beginning for MA(7) and test the model again. In the new model, there are no observable autocorrelations. In case of being correctly selected, the model does not show any patterns in the model residuals. If the residuals are normally distributed, it means that the model correctly fits the series. Tsdisplay() function is applicable in this situation. Residuals plots display a smaller error range, more or less centered around 0. As can be seen, the AIC is smaller when we select (2, 2, 7) for the parameters.

#Fitting the model with new parameters

**>** fit2 **=** arima**(**deseasonal\_open, order**=**c**(**2,2,7**))**

**>** tsdisplay**(**residuals**(**fit2**)**, lag.max**=**15, main**=**'Seasonal Model Residuals'**)**

**>** arima**(**x **=** deseasonal\_open, order **=** c**(**2, 2, 7**))**

Call**:**

arima**(**x **=** deseasonal\_open, order **=** c**(**2, 2, 7**))**

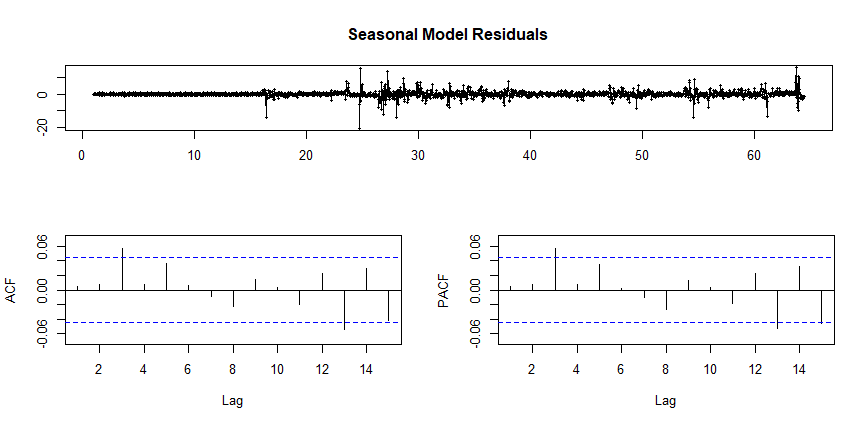
Coefficients**:**

ar1 ar2 ma1 ma2 ma3 ma4 ma5 ma6 ma7

**-**0.1386 **-**0.0933 0.0145 0.0468 0.0161 0.0424 0.0260 0.0326 **-**0.9603

s.e. 0.0244 0.0243 0.0097 0.0095 0.0094 0.0092 0.0088 0.0091 0.0096

sigma**^**2 estimated as 3.863**:** log likelihood **=** **-**4000.62, aic **=** 8021.24

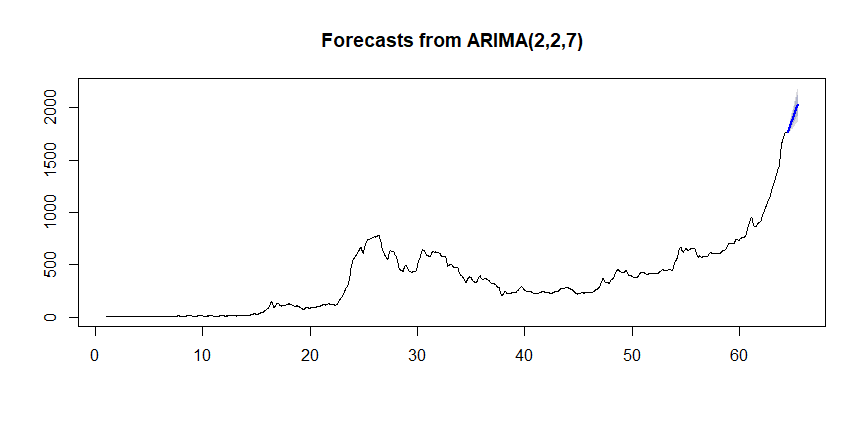


It is easy to do the prediction by a fitted model in R. We just need to determine forecast horizon h periods ahead for making the predictions.

#Forecasting

fcast <- forecast(fit2, h=30)

plot(fcast)



In order to find out how the model will work in future, we can fit the model to a part of the data (hold-out set) and then compare the forecast with the actual values.

#Testing the performance of the model

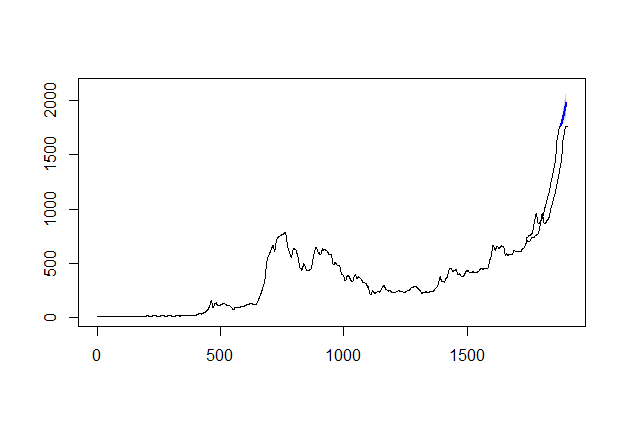
hold <- window(ts(deseasonal\_open), start=1740)

fit\_no\_holdout = arima(ts(deseasonal\_open[-c(1740:1764)]), order=c(2,2,7))

fcast\_no\_holdout <- forecast(fit\_no\_holdout,h=24)

plot(fcast\_no\_holdout, main=" ")

lines(ts(deseasonal\_open))



The blue line that shows forecast has an upward trend which shows the same behavior as the series. This model can be used as a benchmark against more complicated models.