National College of Ireland

Project Submission Sheet – 2015/2016

School of Computing

Student Name:	Sean McNally		
Student ID:	15021581		
Programme:	MSc Data Analytics	**Year:**	2015-2016
Module:	Configuration Manual for Research Project		
Supervisor:	Dr. Jason Roche		
Submission Due Date:	22/08/16		
Project Title:	Predicting the Price of Bitcoin Using Machine Learning		
Word Count:	1200		

Signature: ______________________________

Date: 22/06/2016

Official Use Only	
Signature	
Date:	
Penalty Applied (if applicable)	

User Configuration Manual

1. Introduction:

This user configuration manual will describe the system setup required for the completion of the thesis titled *"Predicting the Price of Bitcoin using Machine Learning"* in terms of both software and hardware. A justification will also be provided as to why certain components were chosen, with reference to other options which they were chosen instead of. In addition, samples of the code listed will be shown and discussed throughout to aid in the reproducibility of the study.

2. Data gathering:

Data relating to the open, high, low and closing price was collected from the Coindesk api. The Coindesk Bitcoin Price Index (BPI) was chosen instead of using data for a specific exchange. Events such as the recent BitFinex hack in which over 190,000BTC was stolen, highlight the risk of focusing on one specific market or exchange. The Coindesk BPI takes an average price across five exchanges (Bitstamp, Bitfinex, Coinbase, itBit and OKCoin).

The data was cleaned using Exel. Feature engineering was also done in Excel.

3. System setup:

Hardware:

Processor: Intel Core I7 6700HQ 2.6GH/z quad core (8 threads)

8GB RAM

250GB SSD and 1TB HDD

GPU: NVIDIA GeForce 940M 2GB

The majority of the research was undertaken running either Windows 10 x64 Home edition or Ubuntu 14.04LTS. This was running on an Intel Core i7 2.6GHz quad core processor, 8GB of RAM, a NVIDIA GeForce 940m 2GB and installed on a 250GB Samsung SSD. An additional 1TB HDD was added to the system to provide separation between the two operating systems. A HDD caddy had to be purchased to fit the HDD into the machine. The caddy holds the HDD and converts the output of the drive to a PCI slot to connect through the former CD drive which has to be removed.

Windows 10 was used for data gathering, extract, transfer and analysis of the data and for further research within the topic area. Zotero was used to keep a centralised database of all relevant papers read. RStudio version 3.2.3 was installed on windows to perform exploratory analysis and to build several machine learning or times series analytical models.

Microsoft Excel 2016 was required for data extraction, transformation and analysis. Microsoft Word and ShareLatex were used for writing drafts and the final paper.

Ubuntu 14.04 was used for all python programming related to the research. Python 2.7 was chosen as the programming language of the more advanced more neural network models. Python was chosen due to the availability of several libraries such as Theano that support machine learning, and recurrent neural networks in particular. Other options here include Torch (Lua language), Caffe (C++) and TensorFlow. Based on a review of the literature, Python was the optimal choice for building recurrent neural networks. In addition, the researcher had some prior experience in python. Theano also offers CUDA optimisation for running python code efficiently on a GPU for speed and performance benefits.

One of the biggest issues faced was getting Theano and CUDA to work with the NVIDIA graphics card driver. The operating system had to be reinstalled numerous times due to the system entering low graphics configuration mode after every install of Theano, CUDA and the Nvidia drivers. The issue here is that the integrated graphics operated off the nouveau graphics driver. When installing the nvidia driver, this driver must be blacklisted prior to the install. This is achieved by entering nouveau.modeset=0 in the GRUB menu for the operating system. Once disabled, the appropriate drivers can be installed correctly. There were dependency issues with the pre-installed drivers so Nvidia driver-358 had to be specifically pip installed.

4. Libraries/ Key software Used:

In both R and python, many packages or libraries were utilised which support machine learning. Originally the RNN was written using Theano. The code contained over 700 lines. The same code ported to Theano is a mere 120 lines for a more advanced model i.e. it has dropout.

4.1 Python/Ubuntu Software Specification:

Python version 2.7

- Python Libraries:
- Keras 1.0.6
- Glances 2.6.2
- Theano 0.8.2
- hyperas 0.2
- hyperopt 0.0.2
- ipython 5
- jupyter 4.3.0
- numpy 1.8.2

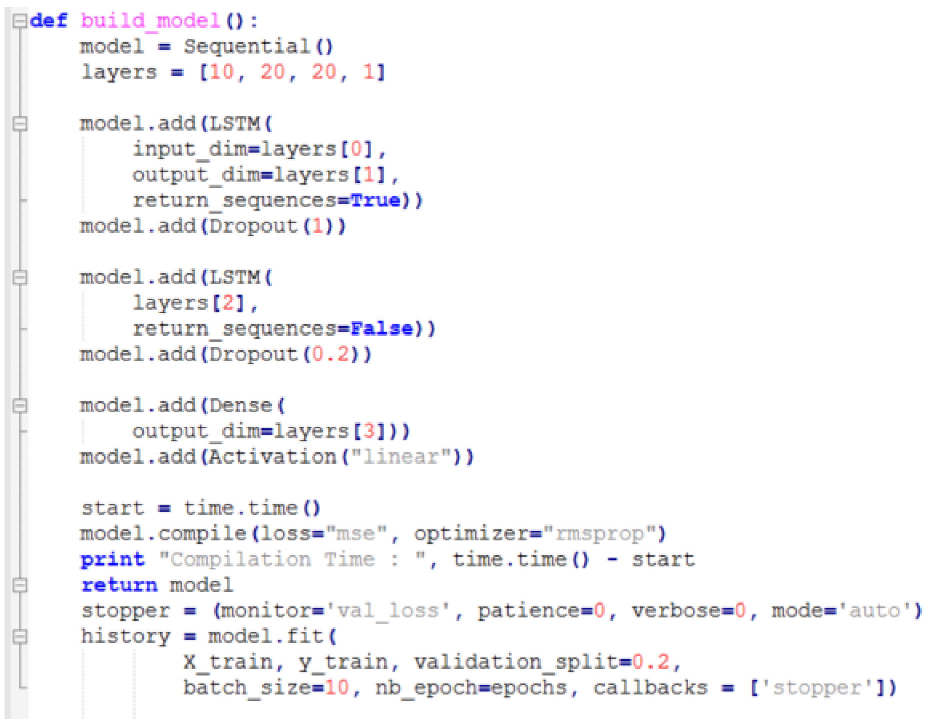
- pandas 0.13.1
- scikit-learn 0.14.1
- scipy 0.13.3
- matplotlib 1.3.1

Ubuntu:

- Cuda 7.5
- Jupyter Notebook
- VirtualEnv
- NVIDIA graphics driver 358
- NVIDIA Prime

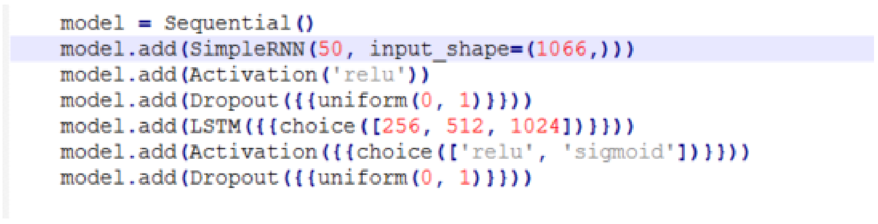
Other libraries listed are required for different Python functions such as timing, producing graphs and validation (scipy, scikit-learn, numpy, matplotlib, pandas) Jupyter/ipython are required for a working notebook environment for testing models. Glances is used to monitor CPU performance while training models. This is useful as it allows you to see how your algorithm is threaded. Hyperopt is a Bayesian optimisation library. Similarly, Hyperas is the same library as a wrapper around Keras. This allows you to optimise all parameter choices of your model in an automated manner. An example of this can be seen in figure 2 below. The versions for all libraries are listed for to avoid potential dependency issues. For Ubuntu CUDA, the NVIDIA driver and Prime are all required to use the GPU. When installing the driver modifications must be made to the grub menu to blacklist the other graphics driver on the system or there are dependency issues and the machine may not boot. If this happens, the NVIDIA drivers must be uninstalled in the tty command line interface. VirtualEnv allows you to contain your work in a virtual environment that is separate from your persistent python installation and libraries. This is beneficial as in the event anything goes wrong a there won't be any system wrong issues. An example of code for the sequential model in Keras is in figure 2 below. The sequential model allows you to stack layers on top of each other.

Figure 1. Keras LSTM



```
def build_model():
    model = Sequential()
    layers = [10, 20, 20, 1]

    model.add(LSTM(
        input_dim=layers[0],
        output_dim=layers[1],
        return_sequences=True))
    model.add(Dropout(1))

    model.add(LSTM(
        layers[2],
        return_sequences=False))
    model.add(Dropout(0.2))

    model.add(Dense(
        output_dim=layers[3]))
    model.add(Activation("linear"))

    start = time.time()
    model.compile(loss="mse", optimizer="rmsprop")
    print "Compilation Time : ", time.time() - start
    return model
    stopper = (monitor='val_loss', patience=0, verbose=0, mode='auto')
    history = model.fit(
            X_train, y_train, validation_split=0.2,
            batch_size=10, nb_epoch=epochs, callbacks = ['stopper'])
```

Figure 2. Hyperas Wrapper



```
model = Sequential()
model.add(SimpleRNN(50, input_shape=(1066,)))
model.add(Activation('relu'))
model.add(Dropout({{uniform(0, 1)}}))
model.add(LSTM({{choice([256, 512, 1024])}}))
model.add(Activation({{choice(['relu', 'sigmoid'])}}))
model.add(Dropout({{uniform(0, 1)}}))
```

4.2 R/Windows Software Specification:

R:

- WaveletComp - Wavelet analysis
- Caret - PCA, correlation, data exploration and decomposition
- Boruta – Feature Selection
- TTR and Forecast – moving average and holt-winters exponential smoothing, auto.arima forecast, autocorrelation function (ACF)

Windows:

- RStudio 3.2.3
- Microsoft Excel and Word 2016
- ShareLatex online Latex compiler for creating the final paper

R is a programming language with a strong emphasis on data analysis, statistics and machine learning. It provides a range of machine learning and statistical packages. One limitation of R is the lack of packages supporting advance deep learning models such as LSTM. As a result, it was used in terms of feature engineering and evaluation.

The WaveComp package was used to decompose Bitcoin closing price using a Haar Wavelet (figure 3). Wavelet Coherence analysis was also performed between closing price and all other independent variables to analyse the correlation of variables on a temporal scale (figure 4). This script installs the WaveletComp package first. Once again closing price is assigned to a data frame, then converted into a time series object. First a discrete Wavelet transform (DWT) is performed on closing price using a Haar wavelet (white.noise). The Wavelet is then reconstructed and plotted.Then the cross Wavelet analysis is performed. Also, na's are removed as the DWT cannot process these. A sample of the script can be seen in figure 5.

Figure 3. Wavelet De-Noised

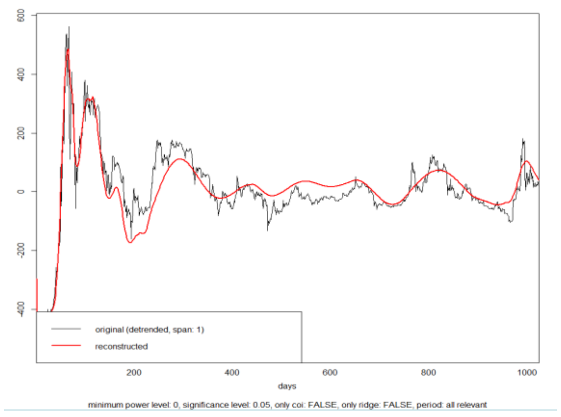


Figure 4.

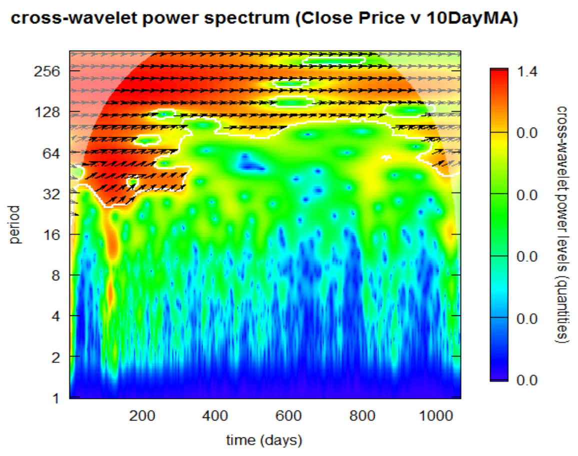
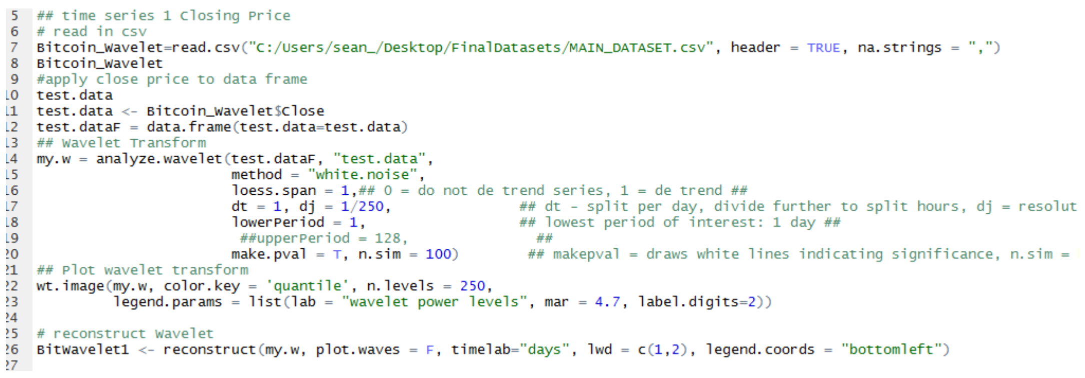


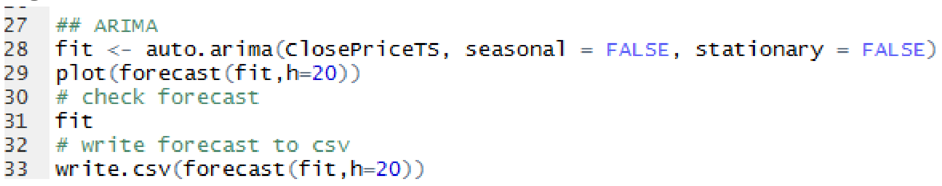
Figure 5. Wavelet



```r
## time series 1 Closing Price
# read in csv
Bitcoin_Wavelet=read.csv("C:/Users/sean_/Desktop/FinalDatasets/MAIN_DATASET.csv", header = TRUE, na.strings = ",")
Bitcoin_Wavelet
#apply close price to data frame
test.data
test.data <- Bitcoin_Wavelet$Close
test.dataF = data.frame(test.data=test.data)
## Wavelet Transform
my.w = analyze.wavelet(test.dataF, "test.data",
                       method = "white.noise",
                       loess.span = 1,## 0 = do not de trend series, 1 = de trend ##
                       dt = 1, dj = 1/250,               ## dt - split per day, divide further to split hours, dj = resolut
                       lowerPeriod = 1,                  ## lowest period of interest: 1 day ##
                        ##upperPeriod = 128,                ##
                       make.pval = T, n.sim = 100)        ## makepval = draws white lines indicating significance, n.sim =
## Plot wavelet transform
wt.image(my.w, color.key = 'quantile', n.levels = 250,
         legend.params = list(lab = "wavelet power levels", mar = 4.7, label.digits=2))

# reconstruct Wavelet
BitWavelet1 <- reconstruct(my.w, plot.waves = F, timelab="days", lwd = c(1,2), legend.coords = "bottomleft")
```

Caret was utilised to perform PCA, correlation, for data exploration in the form of decomposition of a time series (Closing Price). Using the auto.arima function in the forecast package an ARIMA model was fit to five individual splits of the full dataset. Forecasts were then made for the following 20 days out of sample and compared to the actual values. This was used as a benchmark for the neural network models. An example of the ARIMA code can be seen in figure 5 below.

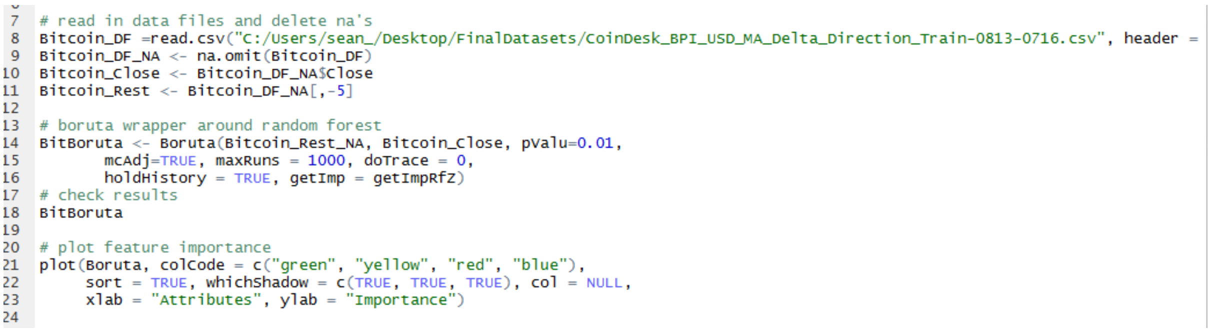
Figure 5. ARIMA Forecast



```r
## ARIMA
fit <- auto.arima(ClosePriceTS, seasonal = FALSE, stationary = FALSE)
plot(forecast(fit,h=20))
# check forecast
fit
# write forecast to csv
write.csv(forecast(fit,h=20))
```

Boruta utilises random forests to evaluate the importance of features. Closing price was the dependent variable. The model returns models accepted or rejected as important to the model based on a p value of .01. This can be seen in figure 6 below.

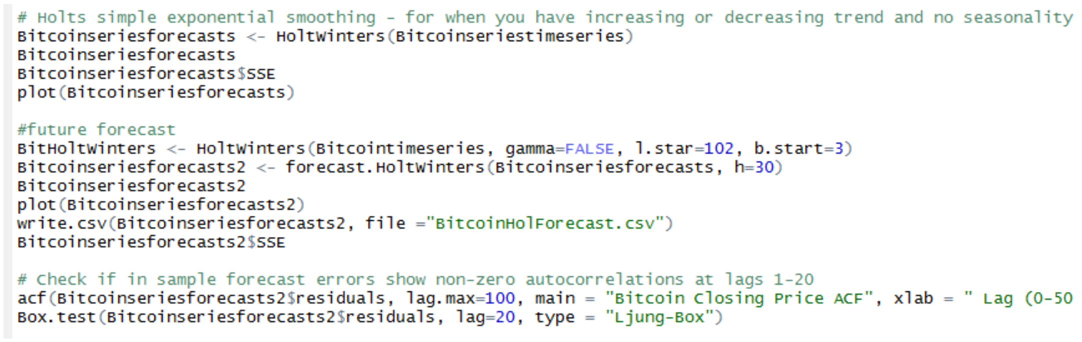
Figure 6. Boruta Feature Selection



```r
# read in data files and delete na's
Bitcoin_DF =read.csv("C:/Users/sean_/Desktop/FinalDatasets/CoinDesk_BPI_USD_MA_Delta_Direction_Train-0813-0716.csv", header =
Bitcoin_DF_NA <- na.omit(Bitcoin_DF)
Bitcoin_Close <- Bitcoin_DF_NA$Close
Bitcoin_Rest <- Bitcoin_DF_NA[,-5]

# boruta wrapper around random forest
BitBoruta <- Boruta(Bitcoin_Rest_NA, Bitcoin_Close, pValu=0.01,
       mcAdj=TRUE, maxRuns = 1000, doTrace = 0,
       holdHistory = TRUE, getImp = getImpRfZ)
# check results
BitBoruta

# plot feature importance
plot(Boruta, colCode = c("green", "yellow", "red", "blue"),
     sort = TRUE, whichShadow = c(TRUE, TRUE, TRUE), col = NULL,
     xlab = "Attributes", ylab = "Importance")
```

TTR and forecast are used to calculate time series forecasts of Bitcoin closing price using the Holt-Winters exponential smoothing. ACF was also calculated to evaluate temporal length (number of days) for the RNN and LSTM based on autocorrelation of closing price and its lag of days in the past and future. The price was also broken down into trend, seasonal and irregular. The trend data was then used as an input for the model. The Holt-Winters exponential smoothing forecast can be seen in figure 7, the ACF lag can be seen in figure 8 and the decomposed time series can be seen in figure 9. As mentioned previously, this de-noised signal was used as input to the final models

Figure 7. Holt-Winters Exponential Smoothing Forecast



```r
# Holts simple exponential smoothing - for when you have increasing or decreasing trend and no seasonality
Bitcoinseriesforecasts <- HoltWinters(Bitcoinseriestimeseries)
Bitcoinseriesforecasts
Bitcoinseriesforecasts$SSE
plot(Bitcoinseriesforecasts)

#future forecast
BitHoltWinters <- HoltWinters(Bitcointimeseries, gamma=FALSE, l.star=102, b.start=3)
Bitcoinseriesforecasts2 <- forecast.HoltWinters(Bitcoinseriesforecasts, h=30)
Bitcoinseriesforecasts2
plot(Bitcoinseriesforecasts2)
write.csv(Bitcoinseriesforecasts2, file ="BitcoinHolForecast.csv")
Bitcoinseriesforecasts2$SSE

# Check if in sample forecast errors show non-zero autocorrelations at lags 1-20
acf(Bitcoinseriesforecasts2$residuals, lag.max=100, main = "Bitcoin Closing Price ACF", xlab = " Lag (0-50 days) ")
Box.test(Bitcoinseriesforecasts2$residuals, lag=20, type = "Ljung-Box")
```

Figure 8. ACF of Bitcoin Closing Price

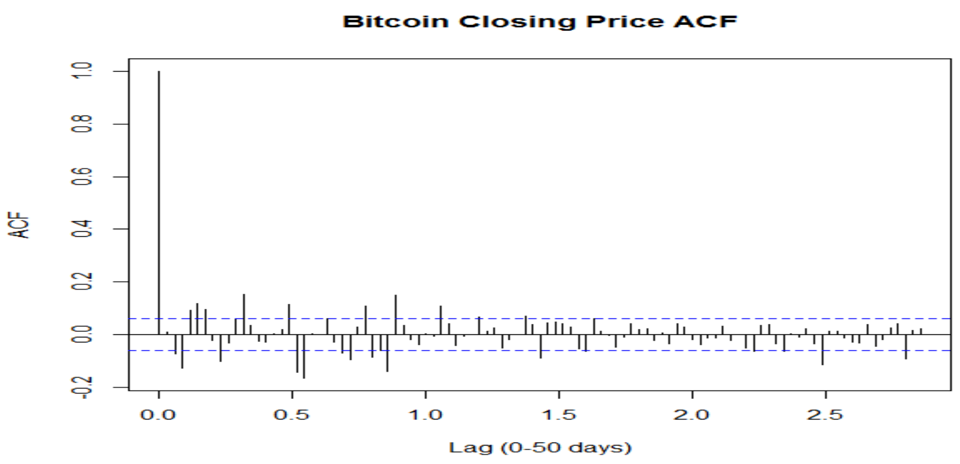
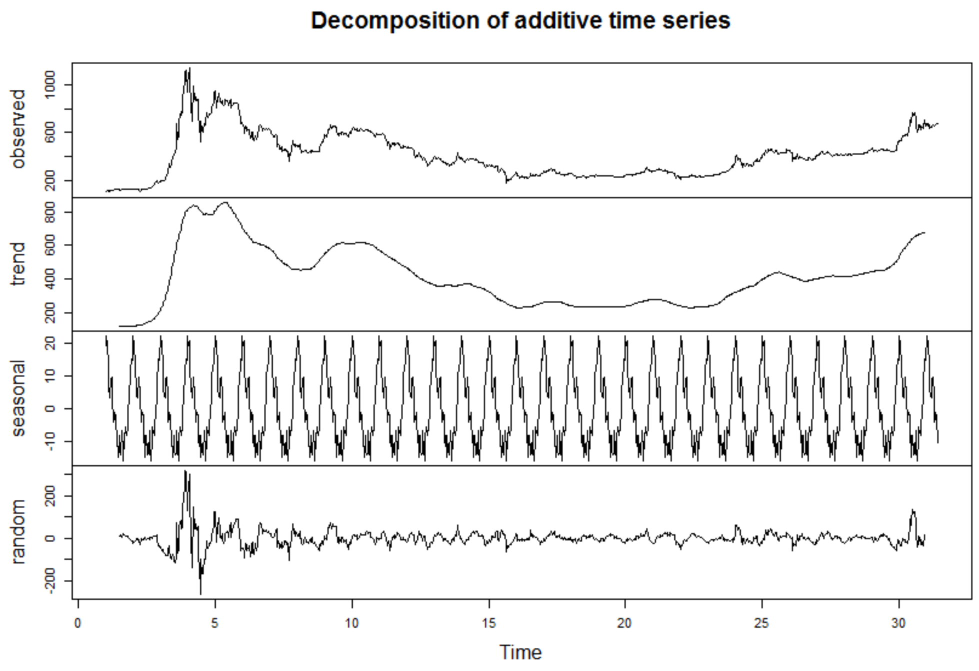


Figure 9. Decomposed Time Series of Bitcoin Closing Price


Decomposition of additive time series
observed
trend
seasonal
random
Time