BITCOIN PRICE

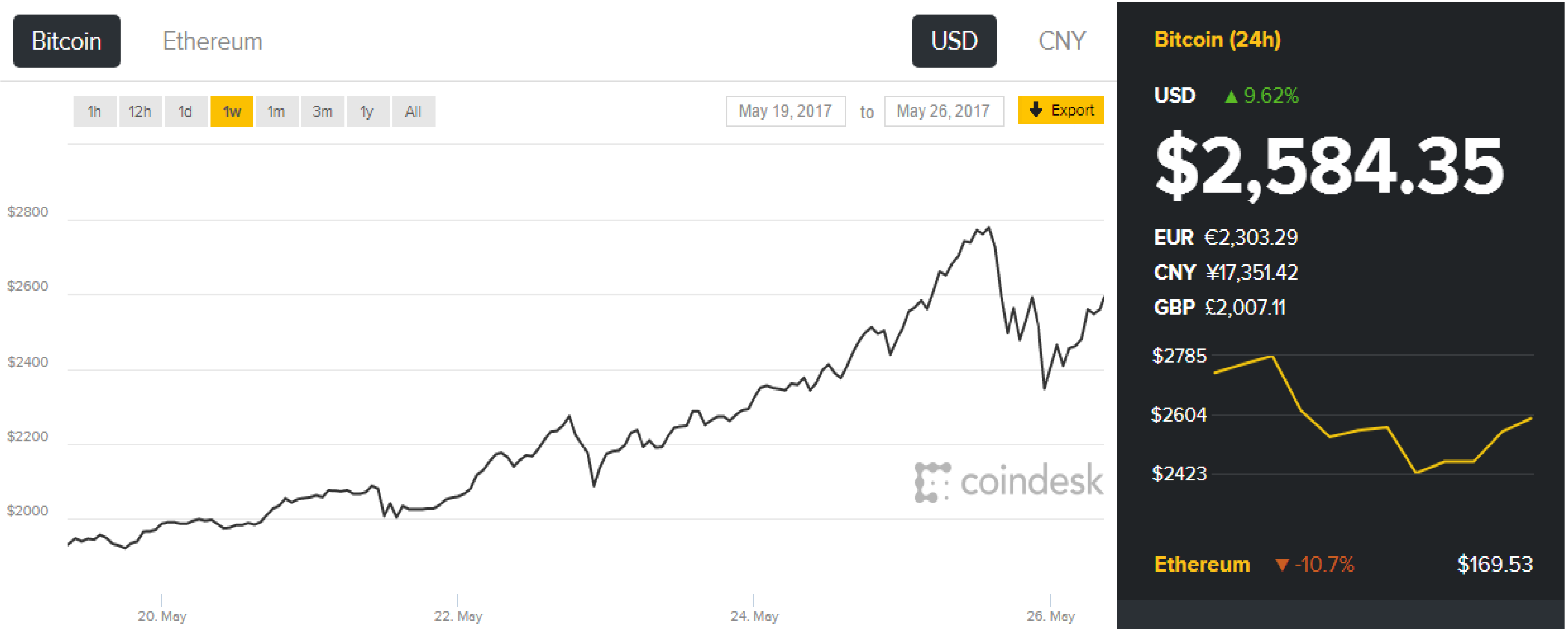
May, 2017

v.1 Narissa Shrestha

THOMSON REUTERS

1. DATA EXPLORATION & TIME SERIES MODEL

Last week vs This week price

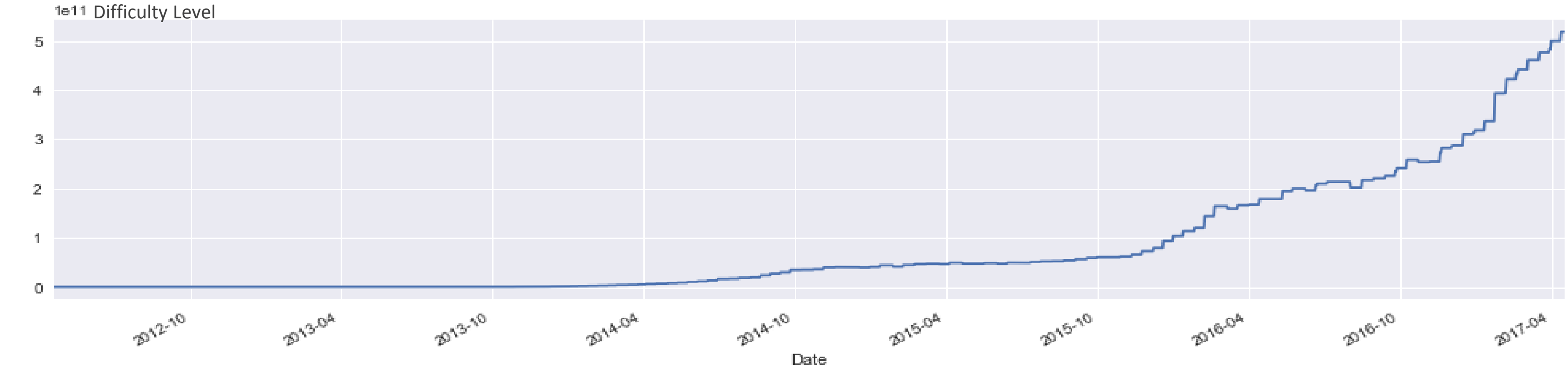


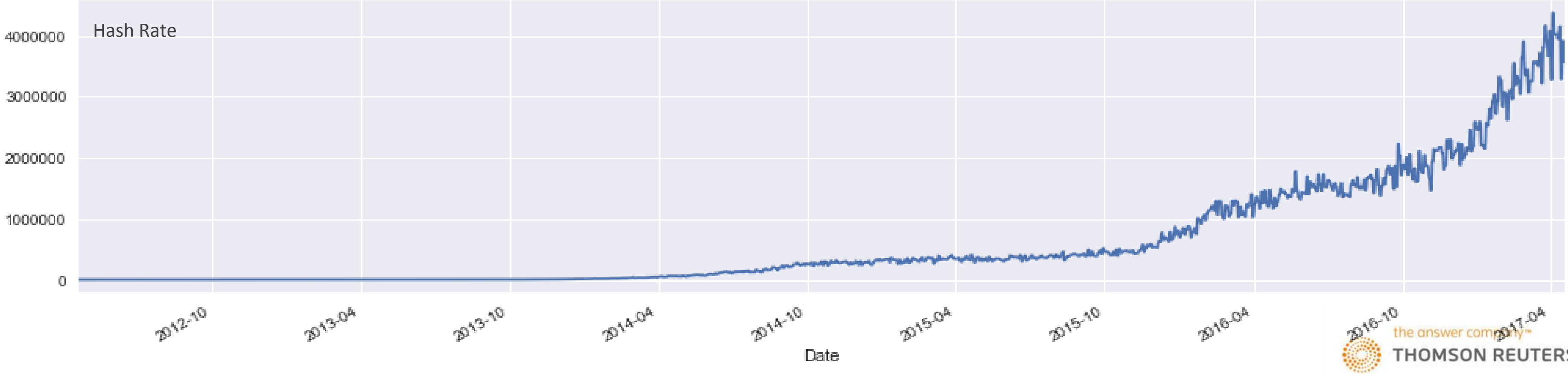
Data Set – Bitcoin Price (2012 – 2017)

- Plot()

- Smoothen, Resample ‘D’, rolling mean, window = 14

Data Set – Difficulty and Hash Rate

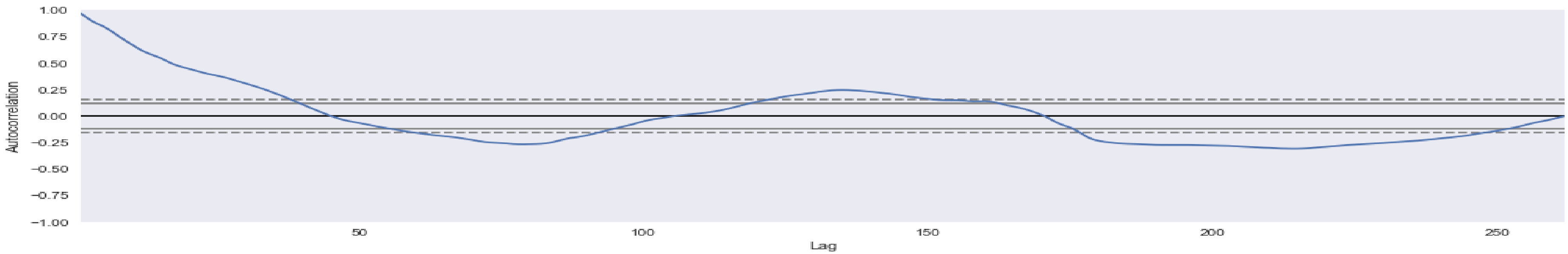


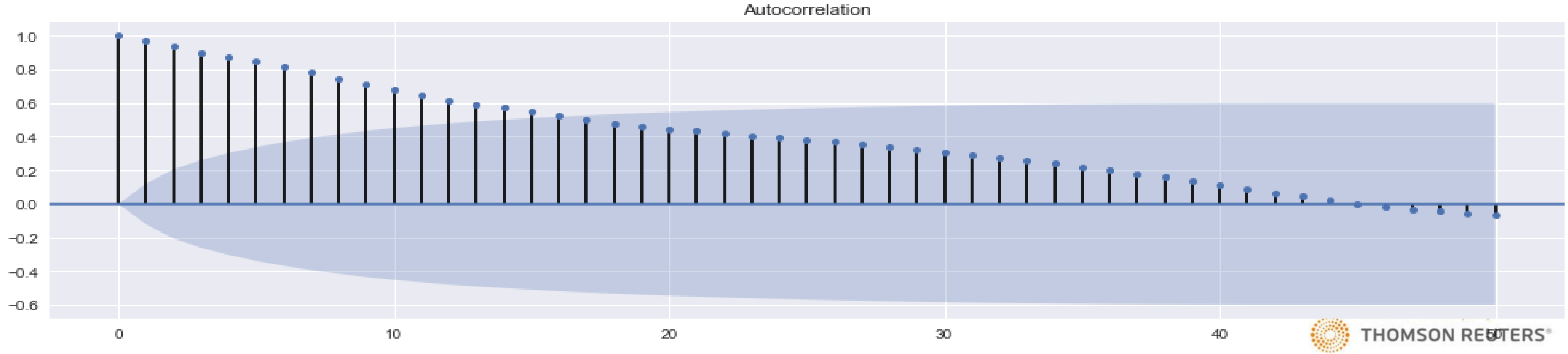


Bitcoin price - Autocorrelation

- Autocorrelation, resample 'D', lag = 1, is 0.99736776309179975
- Autocorrelation, resample 'D', lag = 365, is -0.026137235760321745

- Autocorrelation, resample 'W' lag = 1, is 0.988093713769
- Autocorrelation, resample 'W' lag = 4, is 0.932231852951
- Autocorrelation, resample 'W' lag = 4, is -0.0163816400346





THOMSON REUTERS

Model - ARMA

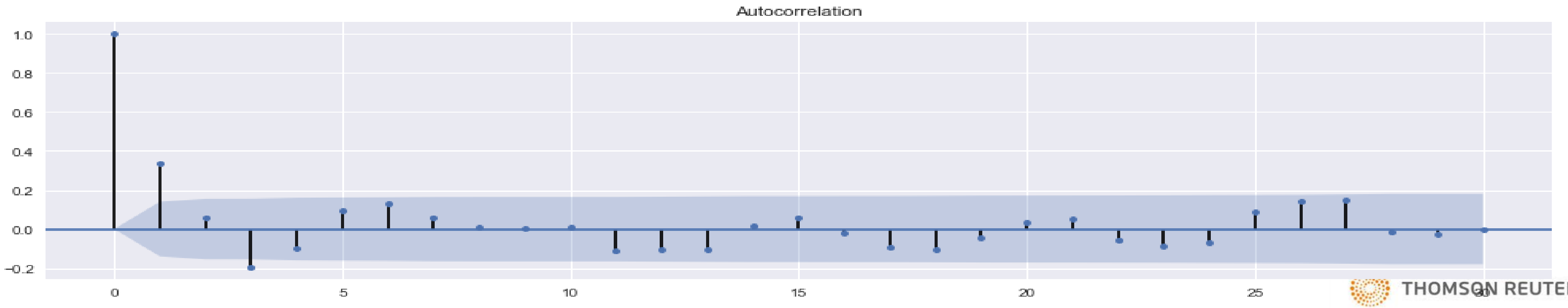
- model = sm.tsa.ARIMA(train, (1, 0, 0)).fit()

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                   Last   No. Observations:                  196
Model:                     ARMA(1, 0)   Log Likelihood               -1026.611
Method:                       css-mle   S.D. of innovations             45.178
Date:                Sat, 13 May 2017   AIC                           2059.221
Time:                        13:20:08   BIC                           2069.056
Sample:                    04-22-2012   HQIC                          2063.203
                         - 01-17-2016
==============================================================================
                 coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const        248.9663    133.193      1.869      0.063       -12.088   510.021
ar.L1.Last     0.9802      0.012     78.599      0.000         0.956     1.005
                                    Roots
=============================================================================
                 Real           Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           1.0202           +0.0000j            1.0202            0.0000
-----------------------------------------------------------------------------
```

Model – ARMA – Plot residual

- model.resid.plot()

- plot_acf(model.resid, lags=30)



THOMSON REUTERS®

Model - ARIMA

- model = sm.tsa.ARIMA(train, (2, 1, 3)).fit()

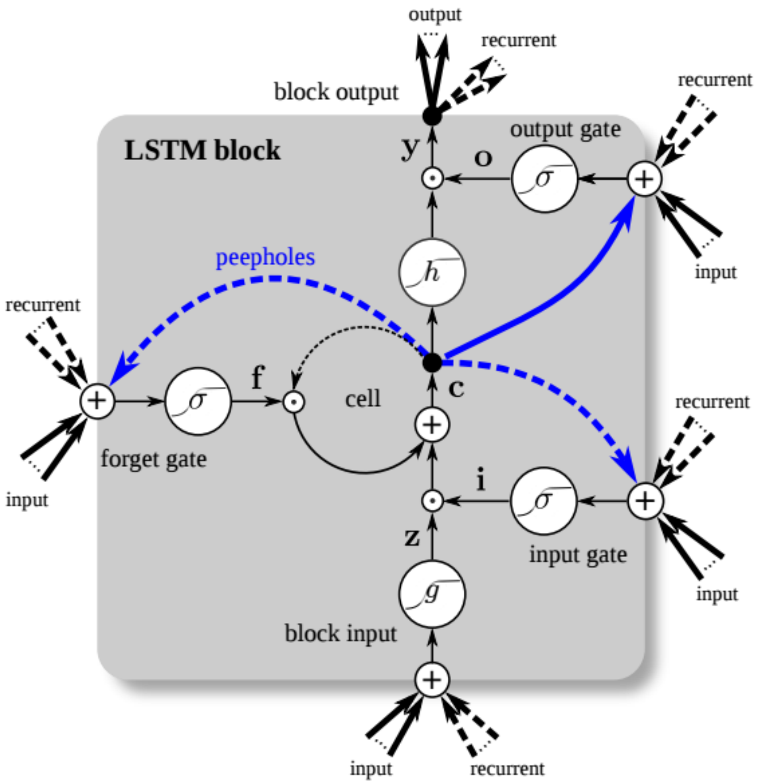
```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                 D.Last   No. Observations:                  195
Model:                 ARIMA(2, 1, 3)   Log Likelihood                -991.570
Method:                       css-mle   S.D. of innovations             39.016
Date:                Sat, 13 May 2017   AIC                           1997.141
Time:                        13:20:09   BIC                           2020.052
Sample:                    04-29-2012   HQIC                          2006.417
                         - 01-17-2016
================================================================================
                   coef    std err          z      P>|z|      [95.0% Conf. Int.]
--------------------------------------------------------------------------------
const            1.9850      3.864      0.514      0.608        -5.588     9.558
ar.L1.D.Last     0.4208      0.139      3.023      0.003         0.148     0.694
ar.L2.D.Last    -0.6223      0.131     -4.751      0.000        -0.879    -0.366
ma.L1.D.Last    -0.0165      0.154     -0.107      0.915        -0.319     0.286
ma.L2.D.Last     0.7041      0.136      5.159      0.000         0.437     0.972
ma.L3.D.Last    -0.0251      0.130     -0.193      0.848        -0.281     0.231
                                    Roots
=============================================================================
                 Real           Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            0.3381           -1.2217j            1.2677           -0.2070
AR.2            0.3381           +1.2217j            1.2677            0.2070
MA.1           -0.0136           -1.1911j            1.1912           -0.2518
MA.2           -0.0136           +1.1911j            1.1912            0.2518
MA.3           28.0560           -0.0000j           28.0560           -0.0000
```

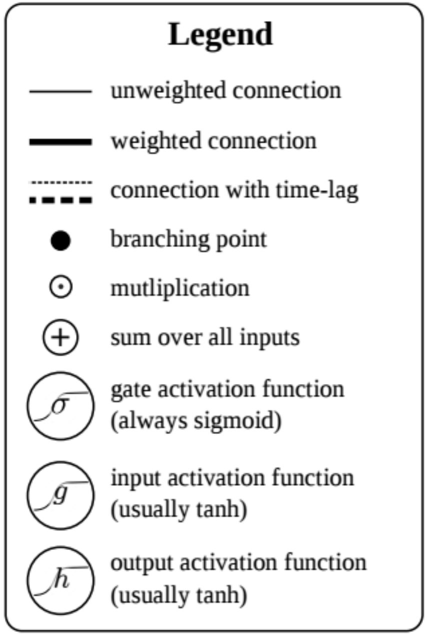
2. LSTM RECURRENT NEURAL NETWORK

Long Short-Term Memory Network

THOMSON REUTERS

LSTM OVERVIEW





LSTMs contain information outside the normal flow of the recurrent network in a gated cell.

- **Forget Gate**: conditionally decides what information to throw away from the block.
- **Input Gate**: conditionally decides which values from the input to update the memory state.
- **Output Gate**: conditionally decides what to output based on input and the memory of the block.

LSTM Network – Key Process

Model required

- from keras.models import Sequential
- from keras.layers import Dense
- from keras.layers import LSTM

1. Define function to create new data set, price at t column and t+1 (next month) to be predicted
2. 'look_back' is number of previous time steps to use as input variables to predict the next period - default to 1
3. Create data set where X is the price at a given time (t) and Y is the price at (t+1)
4. Normalize the dataset, rescale the data to the range of 0-to-1
5. Split the ordered dataset into train and test datasets
6. Create and fit LSTM model
   - network has a visible layer with 1 input, hidden layer with 4 LSTM blocks/neurons
   - output layer makes a single value prediction
   - default sigmoid activation function used for LSTM block
7. Make predictions
8. Invert predictions and calculate MSE
9. Shift train prediction for plotting

```
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
```

```
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

```
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
```

```
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=10, batch_size=1, verbose=2)
```

```
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
```

```
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
```

LSTM Network For Regression – Results

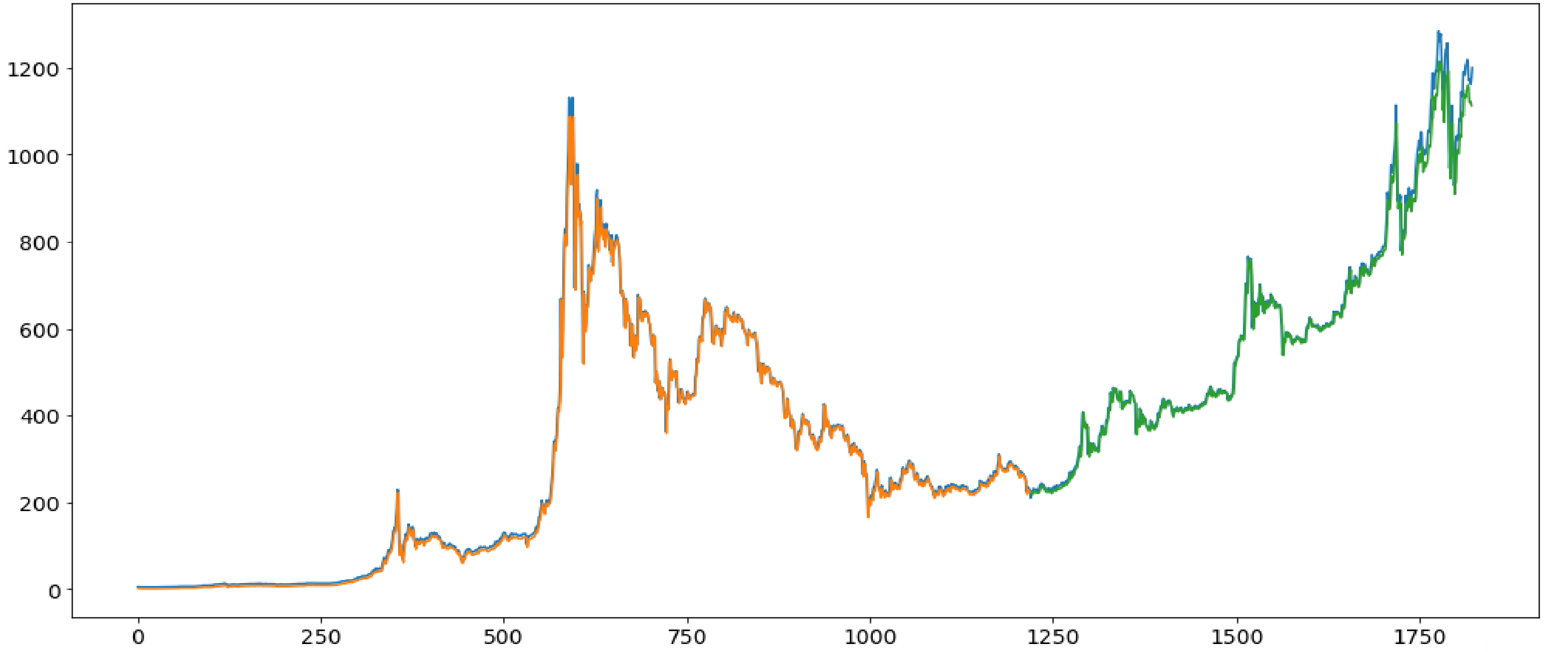
TRAIN SCORE: 21.87 RMSE
TEST SCORE: 22.20 RMSE

Data: 2012 – 2017 price, Epochs = 10, Batch size = 1,
➔ t and t+1 to be predicted

LSTM Network For Regression – Results

TRAIN SCORE: 22.28 RMSE
TEST SCORE: 27.48 RMSE

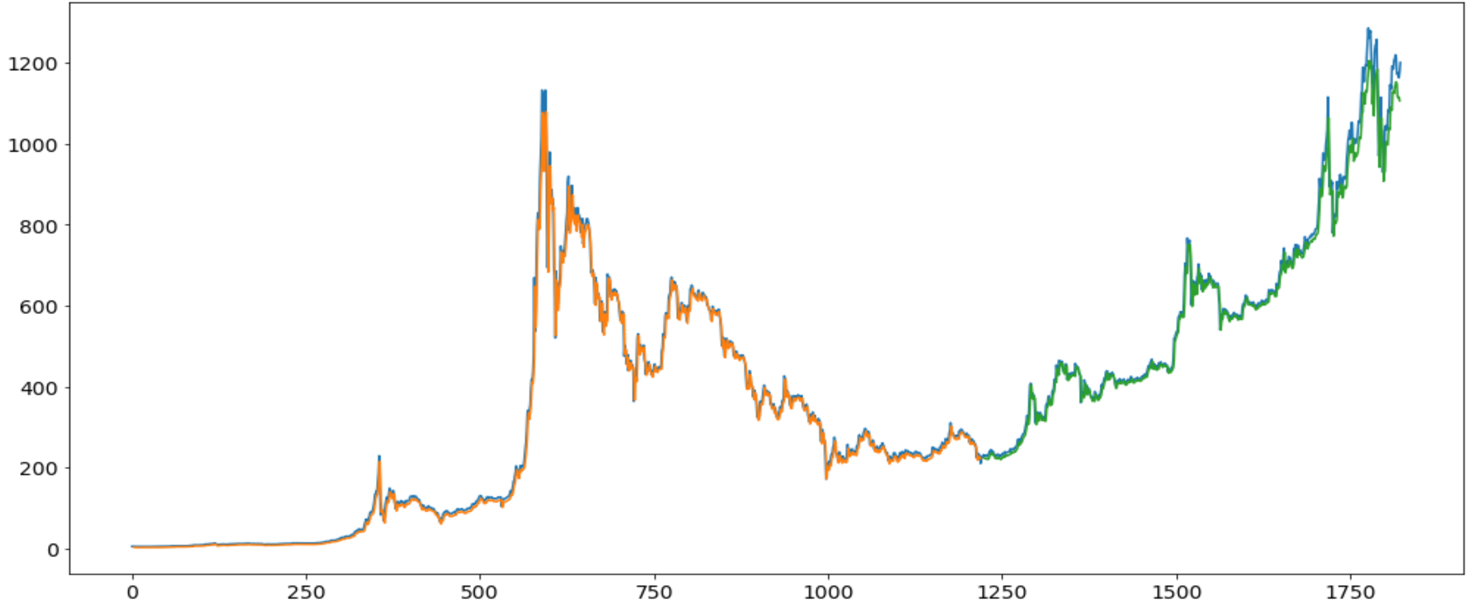
Data: 2012 – 2017 price, Epochs = 300, Batch size = 10
➔ t and t+1 to be predicted



LSTM Network For Regression – Window Method

TRAIN SCORE: 22.65 RMSE
TEST SCORE: 29.86 RMSE

Data: 2012 – 2017 price, Epochs = 100, Batch size = 1
➔ Input t, t-1, t-2 to predict the output variable t+1
➔ Look back = 3

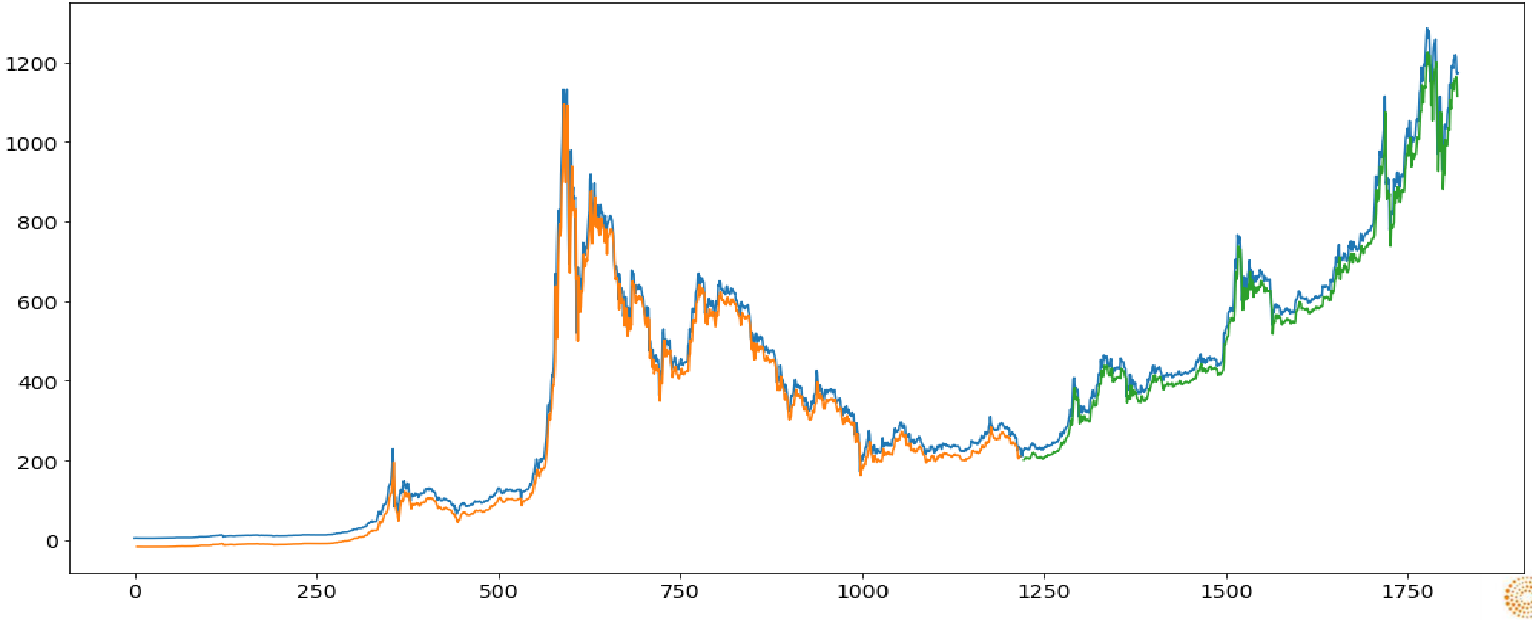


LSTM Network For Regression – Time Steps

TRAIN SCORE: 32.36 RMSE
TEST SCORE: 38.13 RMSE

Data: 2012 – 2017 price, Epochs = 100, Batch size = 1

➔ Input t, t-1, t-2 to predict the output variable t+1
➔ Using the same data representation as in the previous window-based example, except when we reshape the data, we set the columns to be the time steps dimension and change the features dimension back to 1
➔ Reshape input to be [samples, time steps, features]



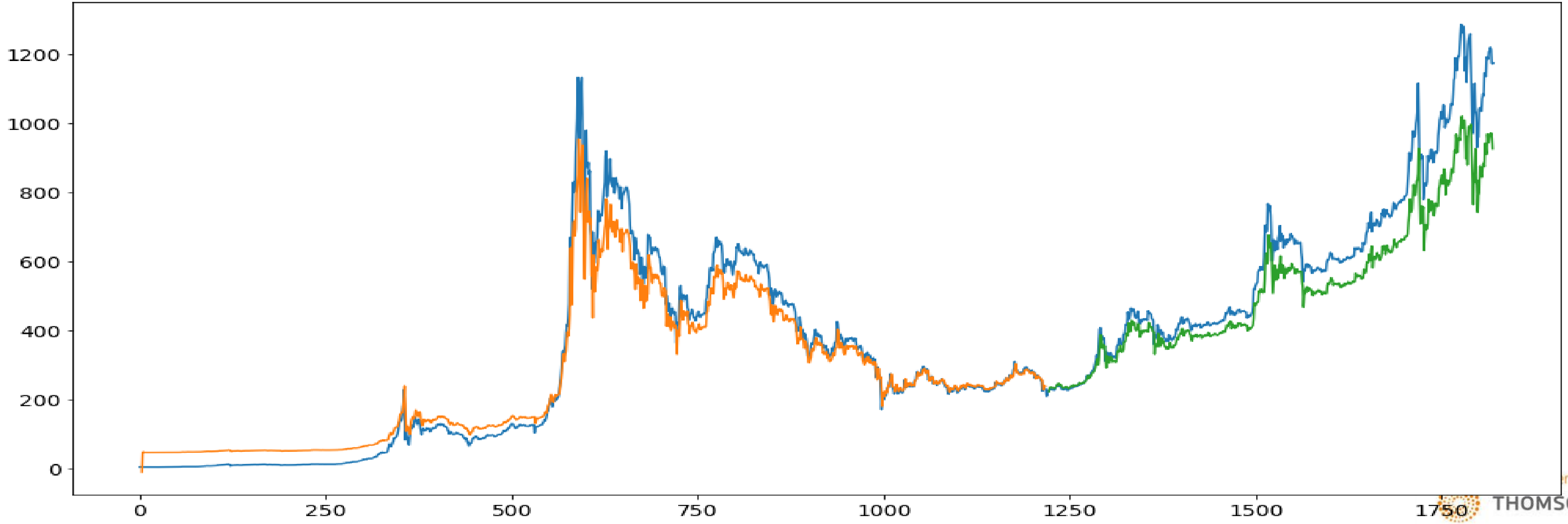
LSTM with Memory Between Batch

TRAIN SCORE: 52.91 RMSE
TEST SCORE: 103.40 RMSE

Data: 2012 – 2017 price

➔ Normally, state within the network is reset after each training batch when fitting the model. We can gain finer control when internal state of LSTM network is cleared in Keras by making the layer "stateful"

```
model = Sequential()
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
for i in range(100):
        model.fit(trainX, trainY, epochs=1, batch_size=batch_size, verbose=2, shuffle=False)
        model.reset_states()
```



Stacked LSTM with Memory Between Batches

TRAIN SCORE: 72.53 RMSE
TEST SCORE: 104.54 RMSE

Data: 2012 – 2017 price,
➔ LSTM networks can be stacked in Keras in the same way that other layer types can be stacked

```
model = Sequential()
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True, return_sequences=True))
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
for i in range(100):
        model.fit(trainX, trainY, epochs=1, batch_size=batch_size, verbose=2, shuffle=False)
        model.reset_states()
```