In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy import stats
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.externals import joblib
from math import sqrt

from random import randint
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import GRU
from keras.callbacks import EarlyStopping
from keras import initializers
from matplotlib import pyplot
from datetime import datetime
from matplotlib import pyplot as plt
import plotly.offline as py
import plotly.graph_objs as go
import apscheduler
from apscheduler.scheduler import Scheduler
import json
import pymongo
from pymongo import MongoClient

from datetime import date
from datetime import datetime
from datetime import timedelta
py.init_notebook_mode(connected=True)
%matplotlib inline

# Start the scheduler
sched = Scheduler()
sched.start()

def create_lookback(dataset, look_back=1):
    X, Y = [], []
    for i in range(len(dataset) - look_back):
        a = dataset[i:(i + look_back), 0]
        X.append(a)
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)

def mean_predict1(price_open,price_open1):
    global initial_mean
    global initial_mean1
    global initial_mean_scalar
    global initial_mean_scalar1
    global X_test
    global X_test1
    initial_mean=np.array(price_open)
    initial_mean1=np.array(price_open1)
    initial_mean_scalar=scaler.transform(initial_mean)
```

```
    initial_mean_scalar1=scaler.transform(initial_mean1)
    X_test = initial_mean_scalar
    X_test1=initial_mean_scalar1
    X_test = np.reshape(X_test, (len(X_test), 1, X_test.shape[1]))
    X_test1=np.reshape(X_test1,(len(X_test1),1,X_test1.shape[1]))
    print(X_test)
    print(X_test1)
    global final_mean
    global final_mean1
    n=8
    mean = model.predict(X_test, verbose=1)
    for i in range(n-1):
        X_test= mean
        X_test = np.reshape(X_test, (len(X_test), 1, X_test.shape[1]))
        yhat = model.predict(X_test, verbose=1)
        mean=yhat
    final_mean=scaler.inverse_transform(mean)
    mean1= model.predict(X_test1, verbose=1)
    for i in range(n-1):
        X_test1= mean1
        X_test1 = np.reshape(X_test1, (len(X_test1), 1, X_test1.shape[1]))
        yhat1 = model.predict(X_test1, verbose=1)
        mean1=yhat1
    final_mean1=scaler.inverse_transform(mean1)
    movement = (final_mean-final_mean1)/price_open
    print(movement)
    global H_pred_movement

    if (movement> 0.0005) :
        H_pred_movement='Rise'

    elif (movement< -0.0005) :
        H_pred_movement='Fall'

    elif (abs(movement)<=0.0005):
        H_pred_movement='Steady'

    else :
        print('invalid value')
```

```
Using TensorFlow backend.
```

In [2]:

```
#def job_function_total():
Client=MongoClient(host=[redacted],port=[redacted])
DB_name=Client['BINANCE']
Collection_15MIN = DB_name['BTC_USD_15MIN']
Client.BINANCE.authenticate("voteAdmin", "voteAdmin")
global data
data = pd.DataFrame(list(Collection_15MIN.find()))
data.drop_duplicates(subset="time_period_start", keep="last",inplace=True)
data=data.tail(1000)
#data=data.tail(2000)
   # print(len(data))
    #data=data[-15000:]
    #data["price_close"]= data["price_close"].astype(float)
data["price_open"]=data["price_open"].astype(float)
    #data["price_high"]=data["price_high"].astype(float)
    #data["price_low"]=data["price_low"].astype(float)
    #data["mean_price"]= (data['price_close']+data['price_high']+data['price_low']+data['price_
data.isnull().values.any()
del data['time_period_end']
del data['volume_traded']
del data['_id']
del data['price_close']
del data['price_high']
del data['price_low']
    #del data['price_open']
del data['time_period_start']
del data['time_period_kr']
a=round(len(data)*0.9)
global train
train=data[:a]
global test
test=data[a:]
working_data = [train,test]
working_data = pd.concat(working_data)
training_set = train.values
training_set = np.reshape(training_set, (len(training_set),1 ))
test_set = test.values
test_set = np.reshape(test_set, (len(test_set),1 ))
    #scale datasets
global scaler
scaler = MinMaxScaler()
training_set = scaler.fit_transform(training_set)
test_set = scaler.transform(test_set)
    #print(test_set)

    # create datasets which are suitable for time series forecasting
look_back = 1
X_train, Y_train = create_lookback(training_set, look_back)
X_test, Y_test = create_lookback(test_set, look_back)

    # reshape datasets so that they will be ok for the requirements of the LSTM model in Keras
X_train = np.reshape(X_train, (len(X_train), 1, X_train.shape[1]))
X_test = np.reshape(X_test, (len(X_test), 1, X_test.shape[1]))

    # initialize sequential model, add 2 stacked LSTM layers and densely connected output neuro
from keras.layers import Dropout, Flatten, Dense, Activation, Reshape, LeakyReLU
from keras.layers import Conv1D, Conv2D
from keras.layers.normalization import BatchNormalization
global model
```

```
60
61  '''
62  model = Sequential()
63  model.add(LSTM(526, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2]), ac
64  model.add(LSTM(526))
65  model.add(Dropout(0.2))
66  model.add(Dense(1))
67  model.add(LeakyReLU())
68
69  '''
70
71  model = Sequential()
72  model.add(LSTM(526, return_sequences=True,input_shape=(X_train.shape[1], X_train.shape[2])))
73  model.add(LSTM(526))
74  model.add(Dense(1))
75
76
77
78      # compile and fit the model
79  model.compile(loss='mean_squared_error', optimizer='adam')
80  history=model.fit(X_train, Y_train, epochs=100,batch_size=180,shuffle=False,
81              validation_data=(X_test, Y_test),
82              callbacks = [EarlyStopping(monitor='val_loss', min_delta=5e-5, patience=20, verbose
83  model.save('Ewha007_BTC.sav')
84  model = load_model('Ewha007_BTC.sav')
85  model._make_predict_function()
86
87  trace1 = go.Scatter(
88      x = np.arange(0, len(history.history['loss']), 1),
89      y = history.history['loss'],
90      mode = 'lines',
91      name = 'Train loss',
92      line = dict(color=('rgb(66, 244, 155)'), width=2, dash='dash')
93  )
94  trace2 = go.Scatter(
95      x = np.arange(0, len(history.history['val_loss']), 1),
96      y = history.history['val_loss'],
97      mode = 'lines',
98      name = 'Test loss',
99      line = dict(color=('rgb(244, 146, 65)'), width=2)
100 )
101
102 data = [trace1, trace2]
103 layout = dict(title = 'Train and Test Loss during training',
104               xaxis = dict(title = 'Epoch number'), yaxis = dict(title = 'Loss'))
105 fig = dict(data=data, layout=layout)
106 py.iplot(fig, filename='training_process')
107
108 # add one additional data point to align shapes of the predictions and true labels
109 X_test = np.append(X_test, scaler.transform(working_data.iloc[-1][0]))
110 X_test = np.reshape(X_test, (len(X_test), 1, 1))
111
112 # get predictions and then make some transformations to be able to calculate RMSE properly in U
113 prediction = model.predict(X_test)
114 prediction_inverse = scaler.inverse_transform(prediction.reshape(-1, 1))
115 Y_test_inverse = scaler.inverse_transform(Y_test.reshape(-1, 1))
116 prediction2_inverse = np.array(prediction_inverse[:,0][1:])
117 Y_test2_inverse = np.array(Y_test_inverse[:,0])
118
119 trace1 = go.Scatter(
120     x = np.arange(0, len(prediction2_inverse), 1),
```

```
    y = prediction2_inverse,
    mode = 'lines',
    name = 'Predicted labels',
    line = dict(color=('rgb(244, 146, 65)'), width=2)
)
trace2 = go.Scatter(
    x = np.arange(0, len(Y_test2_inverse), 1),
    y = Y_test2_inverse,
    mode = 'lines',
    name = 'True labels',
    line = dict(color=('rgb(66, 244, 155)'), width=2)
)

data = [trace1, trace2]
layout = dict(title = 'BITCOIN Comparison of true prices (on the test dataset) with prices our
             xaxis = dict(title = 'Minute'), yaxis = dict(title = 'Price, USD'))
fig = dict(data=data, layout=layout)
py.iplot(fig, filename='results_demonstrating')


#sched.add_cron_job(job_function_total, hour='0,2,4,6,8,10,12,14,16,18,20,22',minute=30,second=
```

```
Train on 899 samples, validate on 99 samples
Epoch 1/100
899/899 [==============================] - 3s 3ms/step - loss: 0.3341 - val_loss:
0.3233
Epoch 2/100
899/899 [==============================] - 0s 67us/step - loss: 0.1897 - val_loss:
0.1327
Epoch 3/100
899/899 [==============================] - 0s 62us/step - loss: 0.0454 - val_loss:
0.0036
Epoch 4/100
899/899 [==============================] - 0s 63us/step - loss: 0.0184 - val_loss:
0.0299
Epoch 5/100
899/899 [==============================] - 0s 58us/step - loss: 0.0445 - val_loss:
0.0089
Epoch 6/100
899/899 [==============================] - 0s 59us/step - loss: 0.0209 - val_loss:
0.0034
```

In [3]:

```
def result():
    pred_hr=datetime.now() - timedelta(hours = 3)
    global result2_time
    result2_time=str(pred_hr.strftime('%Y-%m-%d %H:%M:%S'))
sched.add_cron_job(result, hour='0,2,4,6,8,10,12,14,16,18,20,22')

def result3():
    pred_hr=datetime.now() - timedelta(hours = 3)
    global result3_time
    result3_time=str(pred_hr.strftime('%Y-%m-%d %H:%M:%S'))
sched.add_cron_job(result3, hour='1,3,5,7,9,11,13,15,17,19,21,23', minute=45)

def result4():
    pred_hr=datetime.now() - timedelta(hours = 1)
    global result4_time
    result4_time=str(pred_hr.strftime('%Y-%m-%d %H:%M:%S'))
sched.add_cron_job(result4, hour='0,2,4,6,8,10,12,14,16,18,20,22')

def result5():
    pred_hr=datetime.now() - timedelta(hours = 1)
    global result5_time
    result5_time=str(pred_hr.strftime('%Y-%m-%d %H:%M:%S'))
sched.add_cron_job(result5, hour='1,3,5,7,9,11,13,15,17,19,21,23', minute=45)

def job_function():
    Client=MongoClient(host="[redacted]",port=[redacted]
    DB_name=Client['BINANCE']
    data=DB_name.BTC_USD_15MIN
    Client.BINANCE.authenticate("voteAdmin", "voteAdmin")
    data= pd.DataFrame(list(data.find()))
    data0=data[data.time_period_start==result4_time][-1:]
    print(data0.shape[0])
    if (data0.shape[0]==0):
        data0=data[data.time_period_start==result5_time][-1:]
        data0["price_open"]=data0["price_close"]
    else:
        data0=data0
    print(data0)
    data1=data[data.time_period_start==result2_time][-1:]
    print(data1.shape[0])
    if (data1.shape[0]==0):
        data1=data[data.time_period_start==result3_time][-1:]
        data1["price_open"]=data1["price_close"]
    else:
        data1=data1
    print(data1)
    #data1=pd.DataFrame(data.iloc[[-9]])
    data0["price_open"]=data0["price_open"].astype(float)
    data1["price_open"]=data1["price_open"].astype(float)
    price_open0=np.array(data0["price_open"])
    price_open1=np.array(data1["price_open"])
    price_open0=price_open0.reshape(-1,1)
    price_open1=price_open1.reshape(-1,1)
    mean_predict1(price_open0,price_open1)
sched.add_cron_job(job_function, hour='0,2,4,6,8,10,12,14,16,18,20,22',minute=16)

def time_function():
    pred_hr=datetime.now() + timedelta(hours = 2)
    global prediction_time
```

```
    prediction_time=str(pred_hr.strftime('%Y-%m-%d %H:%M:%S'))
sched.add_cron_job(time_function, hour='0,2,4,6,8,10,12,14,16,18,20,22')

def json_function():
    global BTC_upload_result
    BTC_upload_result=[{'H_nick_name': 'BTC_Ewha007', 'H_model_name': 'Ewha007', 'H_Model_descr
                        'H_pred_time':prediction_time,'H_server_name': 'prediction-herobots003'
                        'H_pred_movement': H_pred_movement}]
sched.add_cron_job(json_function, hour='0,2,4,6,8,10,12,14,16,18,20,22', minute=17)

def upload_result():
    Client=MongoClient(host="[redacted]",port=[redacted])
    DB_name=Client["Ewha007"]
    Collection_A=DB_name.BTC_results
    Client.Ewha007.authenticate("Ewha","Ewha34")
    Collection_A.insert_many(BTC_upload_result)
    Client.close()

#insert BTC json files to Collection_A
sched.add_cron_job(upload_result,hour='0,2,4,6,8,10,12,14,16,18,20,22', minute=18)
```

Out[3]:

```
<Job (name=upload_result, trigger=<CronTrigger (hour='0,2,4,6,8,10,12,14,16,18,20,2
2', minute='18')>)>
```

In [ ]:

```
1
2
3
```