BITCOIN LOW FREQUENCY TRADING STRATEGY USING PRICE, TREND AND SENTIMENT ANALYSIS

ISOM 3350 Group 10

YEUNG, Man Yin Michael (20603418) | CHENG, Lai Him (20596158) | SZE, Kai Tik (20496011) | YAU, Tsz Fung (20512970)

INTRODUCTION

Nowadays, the investment in cryptocurrency has become a hot choice for many investors. In the last 12 months, the cryptocurrency market capitalization increased almost fourfold, reaching $764 billion. The total value of all altcoins apart from bitcoin also appreciated from $60 billion to $225 billion - by more than 270%, showing that more and more investors prefer to choose cryptocurrency as a new type of investment.
Our topic is to apply various models predict the price of cryptocurrency and generate a buy and sell signal. To be specific, we would use three factors in our prediction models:

- Price Prediction (using LSTM)
- Trend Analysis (using the "Turtle Strategy" with a twist)
- Sentiment Analysis on Tweets (using LSTM)

REASONS OF MODEL CHOICES

We believe that historical price behaviors, the historical trend of the prices (breaking n-day resistance or support), and market sentiment on Bitcoins are the major factors affecting the prices.

For Price Prediction, we believe that historical pattern may repeat because they reflects recursive human behavior. We think that we may use past patterns to predict future patterns, which is useful for us to decide on whether to buy or sell, and this buy-sell decision finally affects the total investment return.

REASONS OF MODEL CHOICES

For Trend Analysis, we use "n-day" lookback signals as our major concern, which is actually whether the current price is breaking a "n-day resistance" (n-day rolling max.), or breaking a "n-day support" (n-day rolling min.). We believe that when the investors see such a large-scale increase/decrease break-through, they will adjust their investment decision and thus leading to a good investment opportunity. For example, if investors see a large-scale break-through of resistance, they believe that the Bitcoin price is going to rise further. In that case, as the buying power of Bitcoin increases, the demand and hence price of Bitcoin will rise, which leads to a good investment opportunity. However on the other hand, if it is an n-day support break-through, investors will lose confidence in Bitcoin and thus the demand and price of Bitcoin will drop; and we should end our investment.

REASONS OF MODEL CHOICES

For Sentiment Analysis, it is a comparative new method to analyze the price movement; and also modern machine learning algorithms are required to analysis such a large amount of text data. We believe that Bitcoin, as a kind of financial asset that are publicly available and widely known, has all kinds of individual investors, instead of mainly bank and large-scale financial institutions. As such, the investors may be mainly social media users, expressing opinions on platforms like Tweeter. Tweeter being a large-scale social media platform contains a lot of tweets by netizens - which are possibly Bitcoin investor. So, the sentiment inherited in Bitcoin-related tweets may convey useful insights and market sentiment on Bitcoin when we try to do some deep analysis.

CREATIVITY - TWISTS WE ADOPTED (1)

First, in our project, we not only individually find the results of three individual models, but we also try to combine their results together linearly to seek a even higher return. As the final linear combination of score include all three models as factors, and with a grid search for the best coefficient for linear combination. Theoretically, the result should reflect at least as good as the minimum of the three models. Since using model only is equivalent to "combined model with all weights on one particular model". So, if we do a grid search to find the best model using different coefficients (weights), there should only be enhancement (not any worse than a single individual model.

CREATIVITY - TWISTS WE ADOPTED (2)

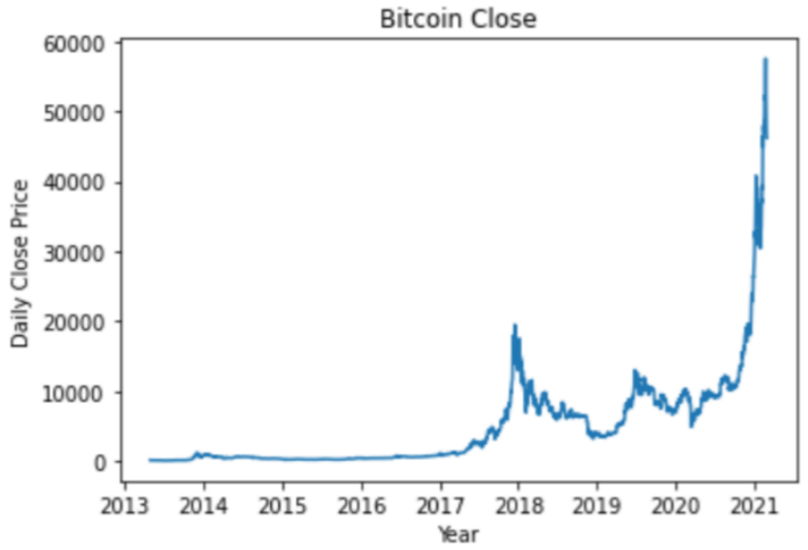
Second, in our trend-following (trend analysis) model. We adopted a weighted sum of different lookback signals instead of a single lookback signal. This is similar to using different day-spans in buy signals and sell signals. In such case, we can improve the original "turtle model" of lookback signals, and seek a not-worse return in the backtest. Grid search always seek better result and not worse than the original result (backtest return). Moreover, the original strategy was only used on stocks, however we try to apply this on Bitcoin - a cryptocurrency. We believe that these two kinds of financial assets has different price behavior and thus there is no guarantee that the model is working. However, as we can see in the backtest later. The model works well!

MAJOR FINDINGS - A SUMMARY

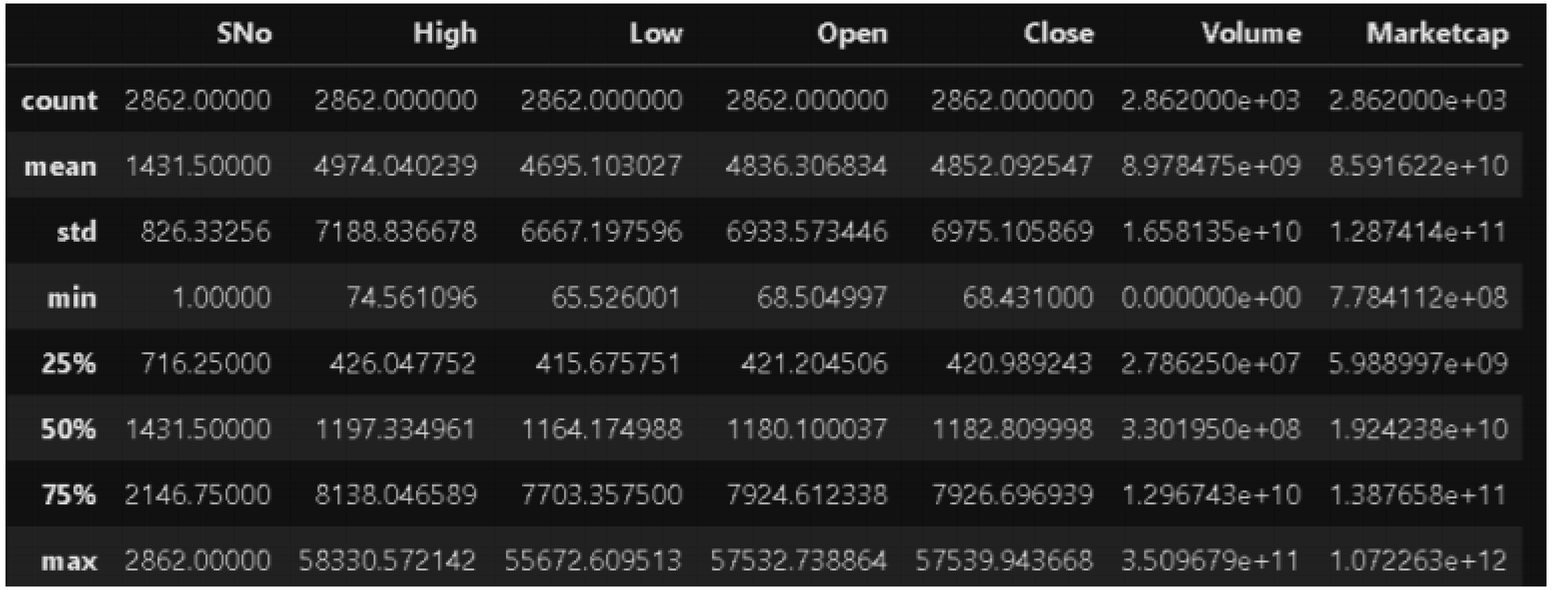
All three models produces a considerable positive profit in the backtest, we believe that Price, Trend, and Sentiment are all important factors in determining future Bitcoin price. Also, the results in the model are decent in general and have no obvious sign of overfitting or irrelevancy. We then try to combine the three models to seek a even better result. The result of mode 4 (combined scores) produces the highest return in our analysis. We thus deduce that it is possible that: when results of different models are combined, there can be enhancement in the results. Also, as the weight in the final result indicates, it seems that sentiment is the most important factor when we try to combine these three different individual models / factors.

DATASETS

First CSV File - 4.8 M BTC price points from Dec 2011 - Mar 2021



Graphical illustration of the dataset

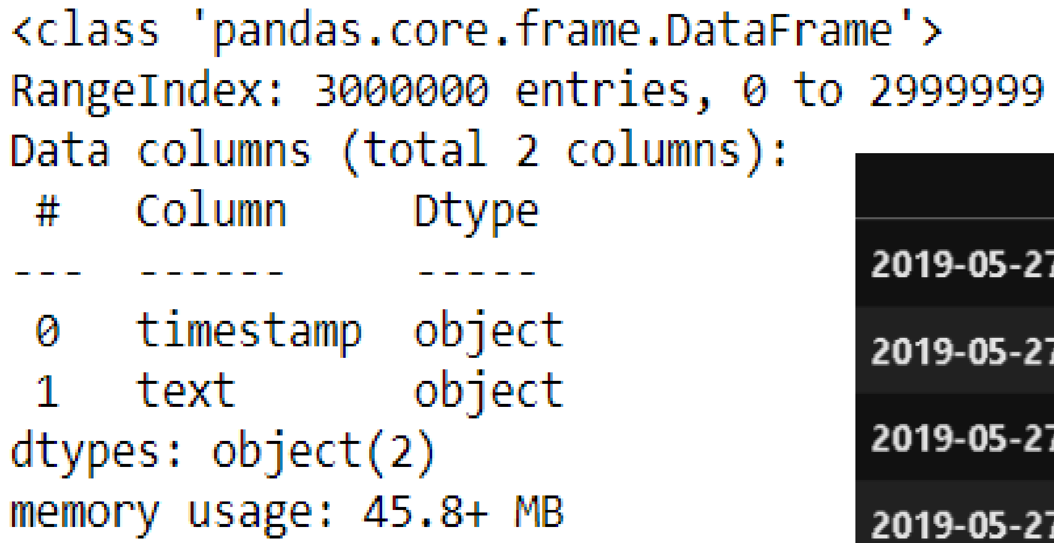


	SNo	High	Low	Open	Close	Volume	Marketcap
count	2862.00000	2862.000000	2862.000000	2862.000000	2862.000000	2.862000e+03	2.862000e+03
mean	1431.50000	4974.040239	4695.103027	4836.306834	4852.092547	8.978475e+09	8.591622e+10
std	826.33256	7188.836678	6667.197596	6933.573446	6975.105869	1.658135e+10	1.287414e+11
min	1.00000	74.561096	65.526001	68.504997	68.431000	0.000000e+00	7.784112e+08
25%	716.25000	426.047752	415.675751	421.204506	420.989243	2.786250e+07	5.988997e+09
50%	1431.50000	1197.334961	1164.174988	1180.100037	1182.809998	3.301950e+08	1.924238e+10
75%	2146.75000	8138.046589	7703.357500	7924.612338	7926.696939	1.296743e+10	1.387658e+11
max	2862.00000	58330.572142	55672.609513	57532.738864	57539.943668	3.509679e+11	1.072263e+12

Descriptive statistics of the dataset

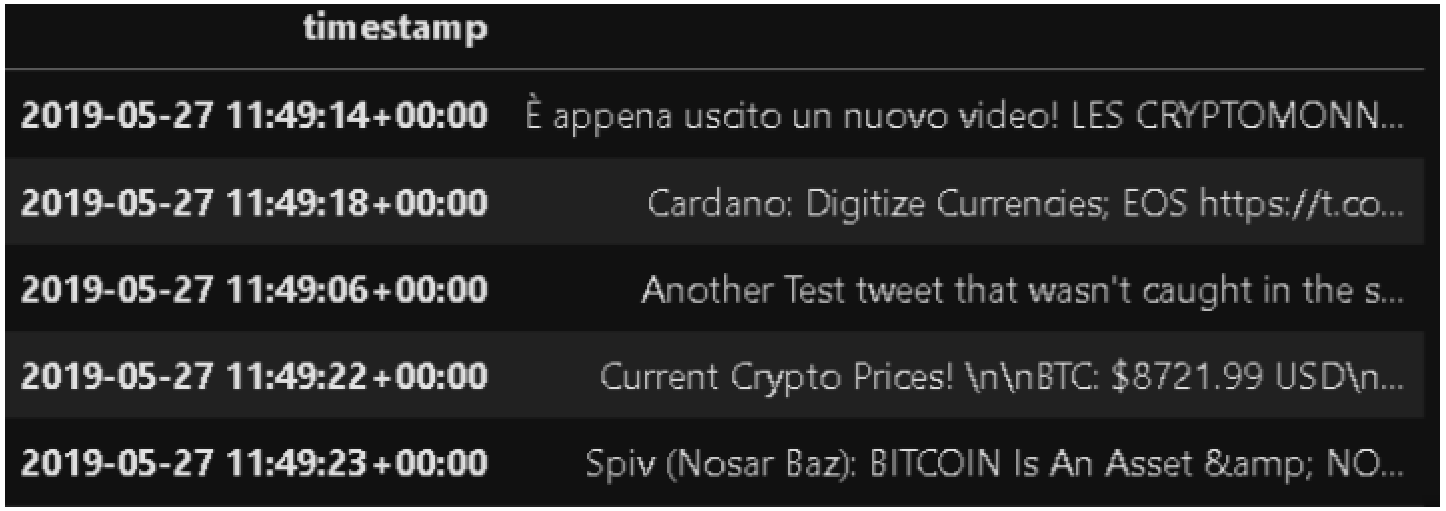
DATASETS

Second CSV File - 3M+ BTC-related Tweets from Jun 2016 - Jan 2019



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000000 entries, 0 to 2999999
Data columns (total 2 columns):
 #   Column     Dtype 
---  ------     ----- 
 0   timestamp  object
 1   text       object
dtypes: object(2)
memory usage: 45.8+ MB
```

Info of dataset



timestamp	
2019-05-27 11:49:14+00:00	È appena uscito un nuovo video! LES CRYPTOMONN...
2019-05-27 11:49:18+00:00	Cardano: Digitize Currencies; EOS https://t.co...
2019-05-27 11:49:06+00:00	Another Test tweet that wasn't caught in the s...
2019-05-27 11:49:22+00:00	Current Crypto Prices! \n\nBTC: $8721.99 USD\n...
2019-05-27 11:49:23+00:00	Spiv (Nosar Baz): BITCOIN Is An Asset &amp; NO...

First few data of the file

METHODOLOGY

Price Prediction	Long Short-term Memory (LSTM)
Sentiment Analysis	Long Short-term Memory (LSTM)
Trend-following Strategy	Lookback Signals Combinations
Prediction Model	Linear Combination of 3 Models' Result

BASIC IDEA OF LSTM

LSTM is powerful in sequence prediction with the ability to store past information,
showing previous prices are crucial for prediction

We believe the past patterns in Price and Sentiment are both useful for predicting
Future prices and thus useful for generating Buy/Sell signals, which the ultimate
Goal is to seek higher investment return (which will be maximized if you know everything
Including Bitcoin prices In the future)

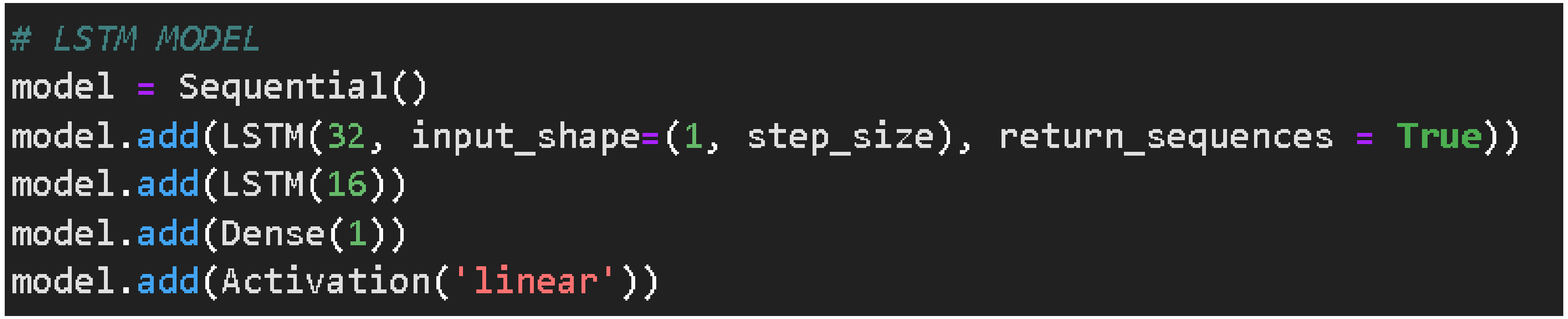
PRICE PREDICTION USING LSTM - MODEL

For full codes of this model (Part I), please refer to these files:

part_1_price_prediction_lstm.ipynb   preprocessing.py   return_calculation.py

We will skip the preprocessing of data as it is not our major concern.

Our Model: A Sequential LSTM model performing the followings

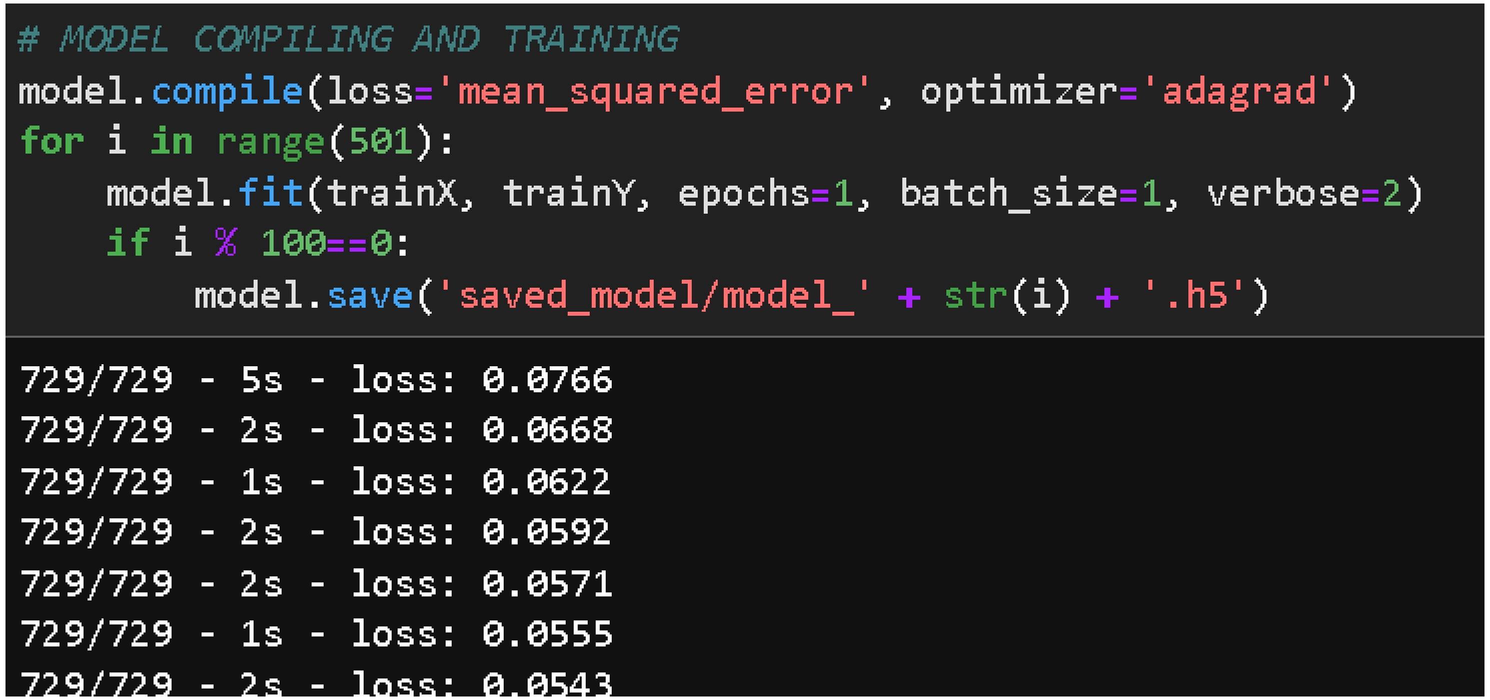


```
# LSTM MODEL
model = Sequential()
model.add(LSTM(32, input_shape=(1, step_size), return_sequences = True))
model.add(LSTM(16))
model.add(Dense(1))
model.add(Activation('linear'))
```

PRICE PREDICTION USING LSTM - TRAINING THE MODEL

We then train our model with the training data (not testing data).

We also save our model so it be used again later.

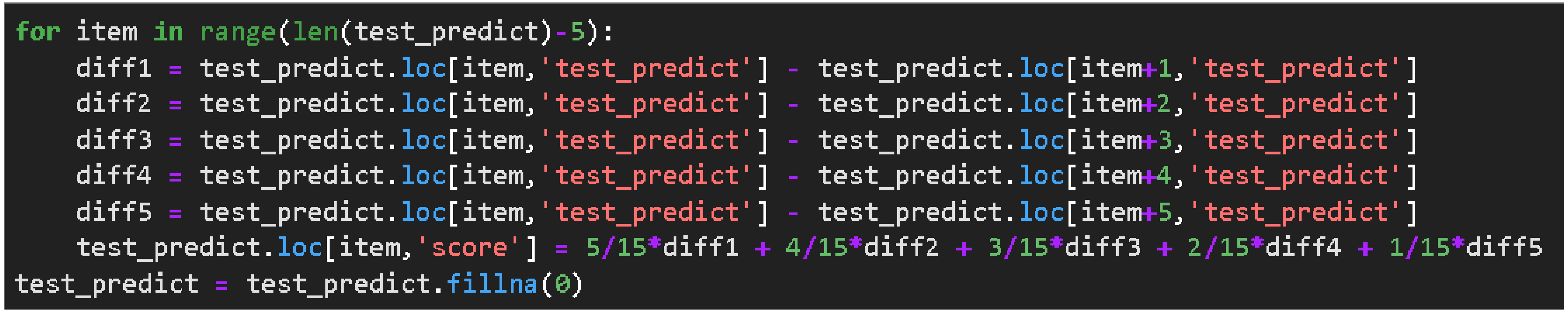


```
# MODEL COMPILING AND TRAINING
model.compile(loss='mean_squared_error', optimizer='adagrad')
for i in range(501):
    model.fit(trainX, trainY, epochs=1, batch_size=1, verbose=2)
    if i % 100==0:
        model.save('saved_model/model_' + str(i) + '.h5')
```

```
729/729 - 5s - loss: 0.0766
729/729 - 2s - loss: 0.0668
729/729 - 1s - loss: 0.0622
729/729 - 2s - loss: 0.0592
729/729 - 2s - loss: 0.0571
729/729 - 1s - loss: 0.0555
729/729 - 2s - loss: 0.0543
```

PRICE PREDICTION USING LSTM - CALCULATIONS

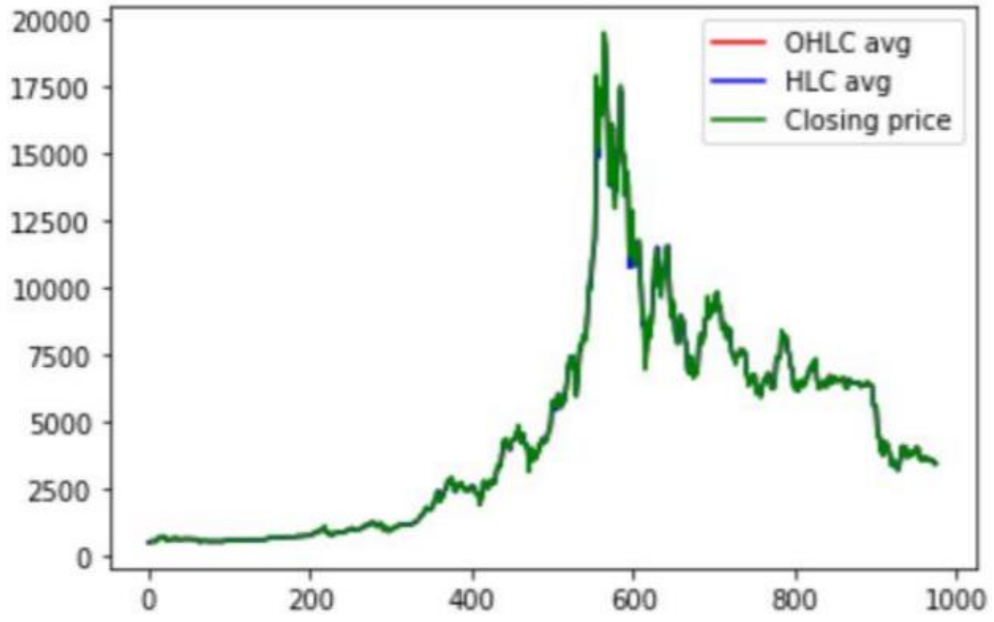
After we get the predicted price, we calculate the difference of the predicted value at the specific day with the five following days, and calculate the weighted average with the weightings 5/15, 4/15, 3/15, 2/15, 1/15 respectively as we emphasize on the price change of the nearer future (this can be further improved to be "trained weights", but we keep them as fixed values for simplicity and saving runtime).



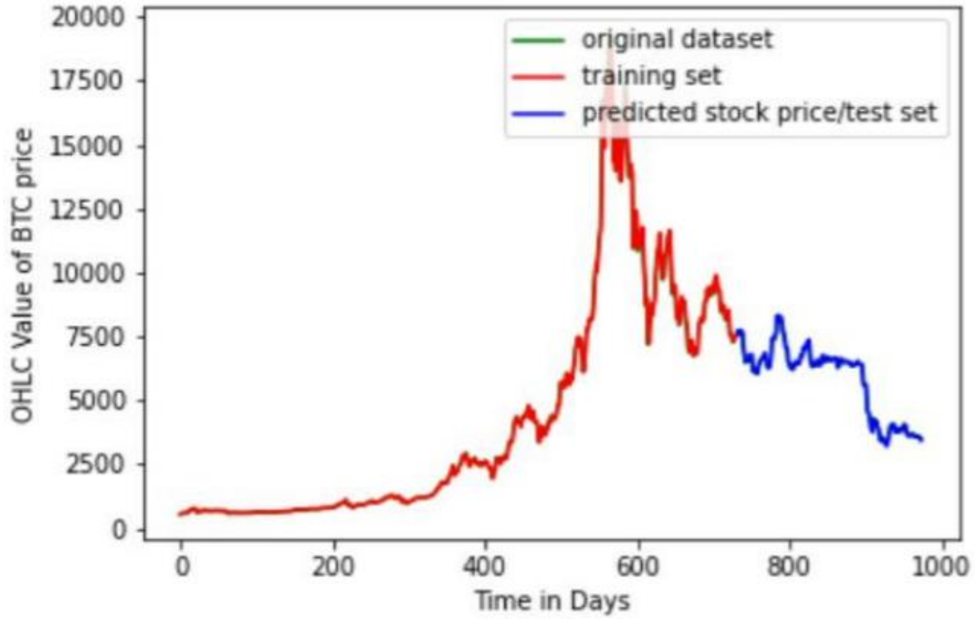
```
for item in range(len(test_predict)-5):
    diff1 = test_predict.loc[item,'test_predict'] - test_predict.loc[item+1,'test_predict']
    diff2 = test_predict.loc[item,'test_predict'] - test_predict.loc[item+2,'test_predict']
    diff3 = test_predict.loc[item,'test_predict'] - test_predict.loc[item+3,'test_predict']
    diff4 = test_predict.loc[item,'test_predict'] - test_predict.loc[item+4,'test_predict']
    diff5 = test_predict.loc[item,'test_predict'] - test_predict.loc[item+5,'test_predict']
    test_predict.loc[item,'score'] = 5/15*diff1 + 4/15*diff2 + 3/15*diff3 + 2/15*diff4 + 1/15*diff5
test_predict = test_predict.fillna(0)
```

PRICE PREDICTION - PREDICTIONS

The original time series of Bitcoin price from 2017 to 2019



The training and test result for Bitcoin price from 2017. to 2019

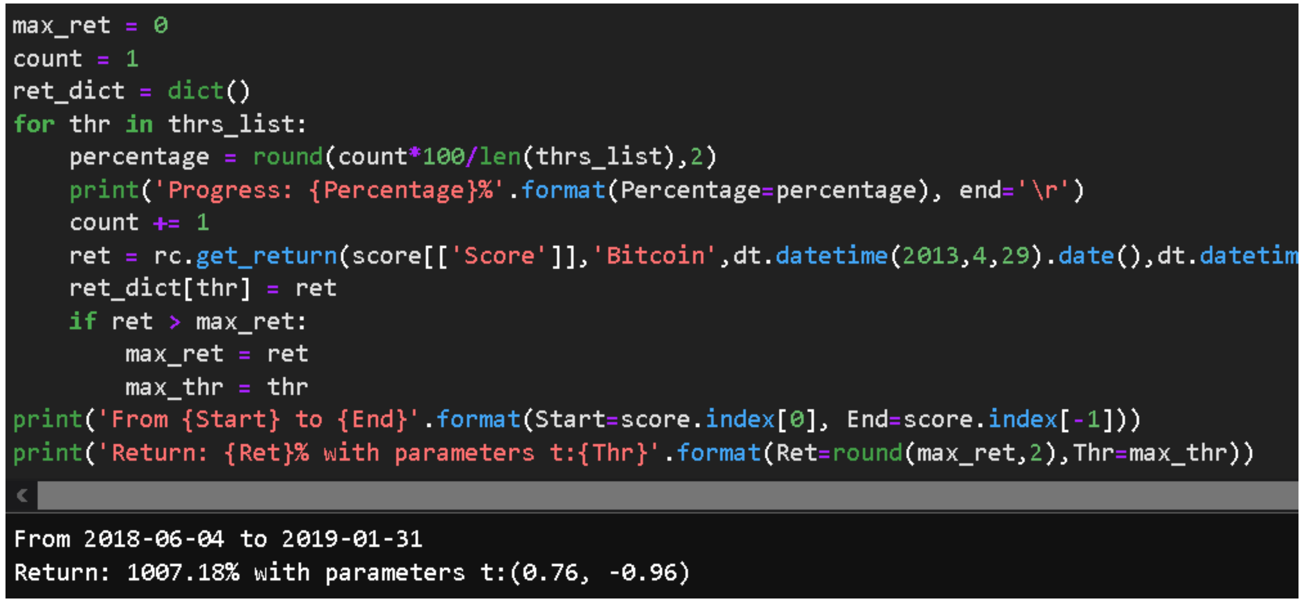


The model works extremely well and generate decent amount of return (1007% in 242 days = 3638% p.a.)

Note: Bitcoin investment value is expected to be increasing 37-fold per year, which is quite abnormal.

PRICE PREDICTION USING LSTM - CALCULATE RETURN

We finally calculate the return based on the calculation above. The calculation of return uses a helper function in return_calculation.py . This function can be used for multiple models. We also implement a grid search for the best parameters and thresholds.

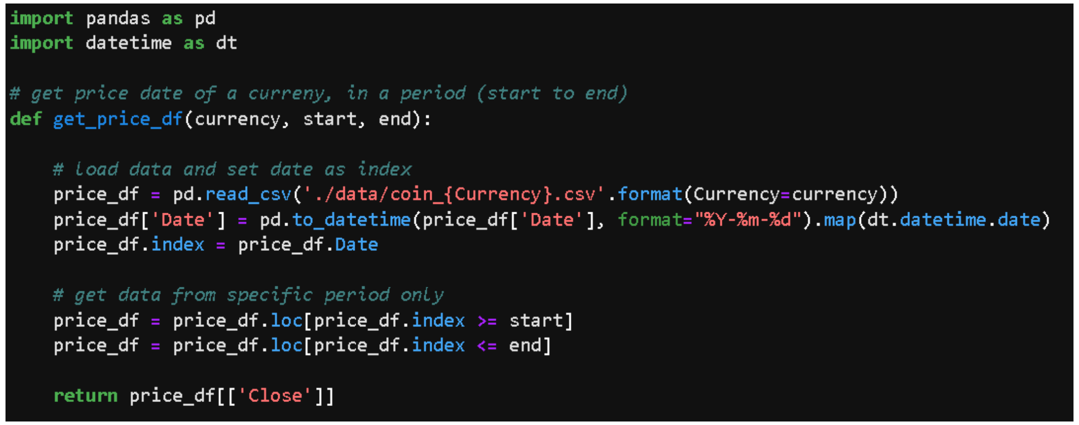


```
max_ret = 0
count = 1
ret_dict = dict()
for thr in thrs_list:
    percentage = round(count*100/len(thrs_list),2)
    print('Progress: {Percentage}%'.format(Percentage=percentage), end='\r')
    count += 1
    ret = rc.get_return(score[['Score']],'Bitcoin',dt.datetime(2013,4,29).date(),dt.datetim
    ret_dict[thr] = ret
    if ret > max_ret:
        max_ret = ret
        max_thr = thr
print('From {Start} to {End}'.format(Start=score.index[0], End=score.index[-1]))
print('Return: {Ret}% with parameters t:{Thr}'.format(Ret=round(max_ret,2),Thr=max_thr))
```

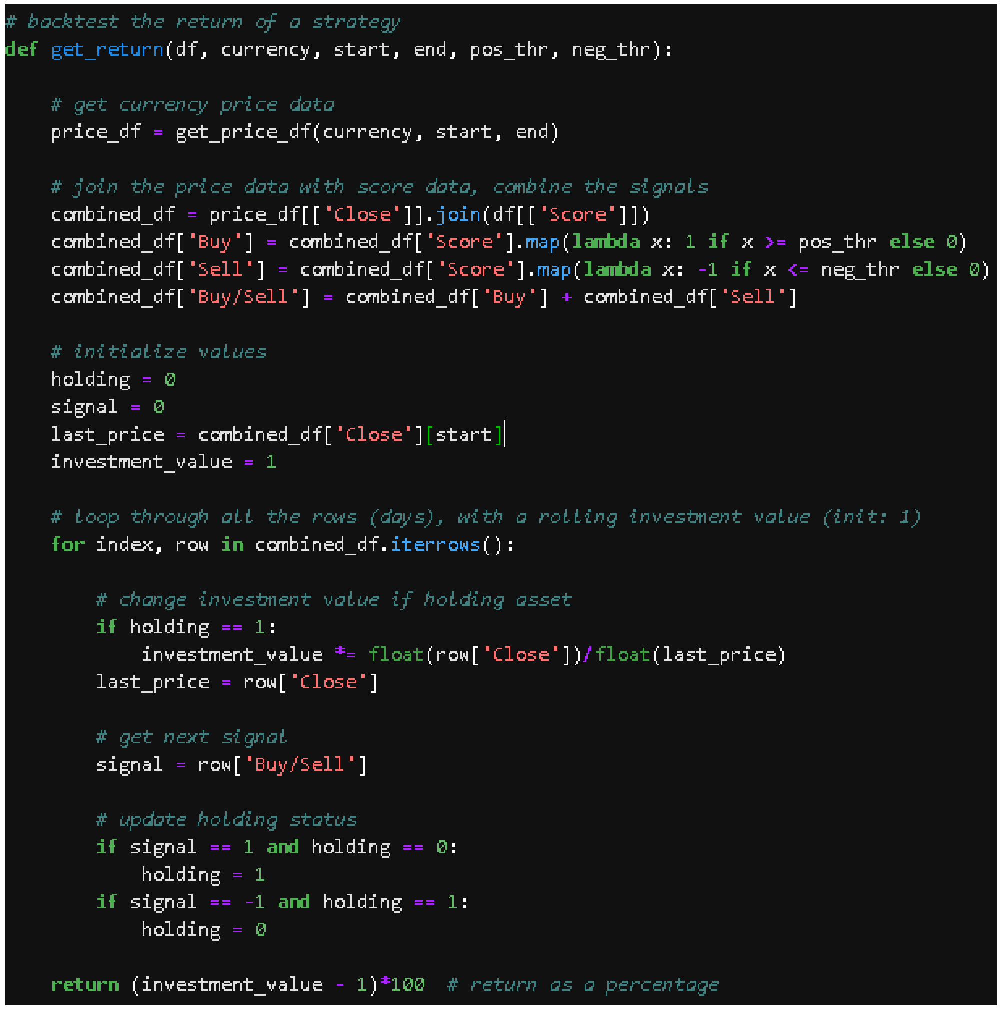
```
From 2018-06-04 to 2019-01-31
Return: 1007.18% with parameters t:(0.76, -0.96)
```

HELPER FUNCTION IN RETURN_CALCULATION.PY

This helper function can loop through a desired period to output a total investment return given a column of scores. We will explain this in details later.



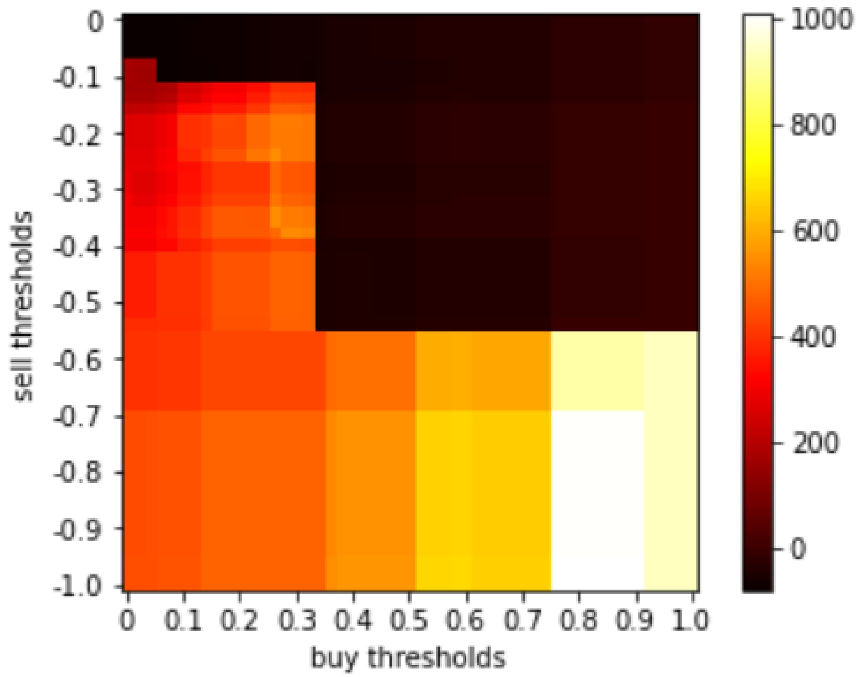
```python
import pandas as pd
import datetime as dt

# get price date of a curreny, in a period (start to end)
def get_price_df(currency, start, end):

    # load data and set date as index
    price_df = pd.read_csv('./data/coin_{Currency}.csv'.format(Currency=currency))
    price_df['Date'] = pd.to_datetime(price_df['Date'], format="%Y-%m-%d").map(dt.datetime.date)
    price_df.index = price_df.Date

    # get data from specific period only
    price_df = price_df.loc[price_df.index >= start]
    price_df = price_df.loc[price_df.index <= end]

    return price_df[['Close']]
```

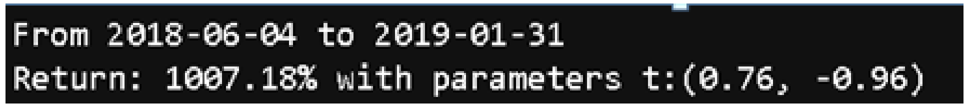


```python
# backtest the return of a strategy
def get_return(df, currency, start, end, pos_thr, neg_thr):

    # get currency price data
    price_df = get_price_df(currency, start, end)

    # join the price data with score data, combine the signals
    combined_df = price_df[['Close']].join(df[['Score']])
    combined_df['Buy'] = combined_df['Score'].map(lambda x: 1 if x >= pos_thr else 0)
    combined_df['Sell'] = combined_df['Score'].map(lambda x: -1 if x <= neg_thr else 0)
    combined_df['Buy/Sell'] = combined_df['Buy'] + combined_df['Sell']

    # initialize values
    holding = 0
    signal = 0
    last_price = combined_df['Close'][start]
    investment_value = 1

    # loop through all the rows (days), with a rolling investment value (init: 1)
    for index, row in combined_df.iterrows():

        # change investment value if holding asset
        if holding == 1:
            investment_value *= float(row['Close'])/float(last_price)
        last_price = row['Close']

        # get next signal
        signal = row['Buy/Sell']

        # update holding status
        if signal == 1 and holding == 0:
            holding = 1
        if signal == -1 and holding == 1:
            holding = 0

    return (investment_value - 1)*100  # return as a percentage
```

PRICE PREDICTION USING LSTM - GRID SEARCH

The results of the grid search is then summarized and visualized. The area with the lightest color indicates the thresholds yielding the highest return. We can see that more polarized thresholds (i.e., thresholds with a greater magnitude) generate a higher return in general. The best pairs of thresholds are located in the white rectangle in the right-bottom corner





```
From 2018-06-04 to 2019-01-31
Return: 1007.18% with parameters t:(0.76, -0.96)
```

TREND-FOLLOWING STRATEGY – A SINGLE SIGNAL

For full codes of this model (Part II), please refer to these files:

part_2_trend_following.ipynb return_calculation.py

An original trend-following trading model was first introduced by Richard Dennis et al. back in 1983 as the "Turtle Experiment", and has been widely recognized as a tremendous success. We simulated this model in our Bitcoin prices backtest, with a twist to adopt a weighted average of different lookback signals. The "score" for a single look-back signal at time t can be mathematically expressed as:

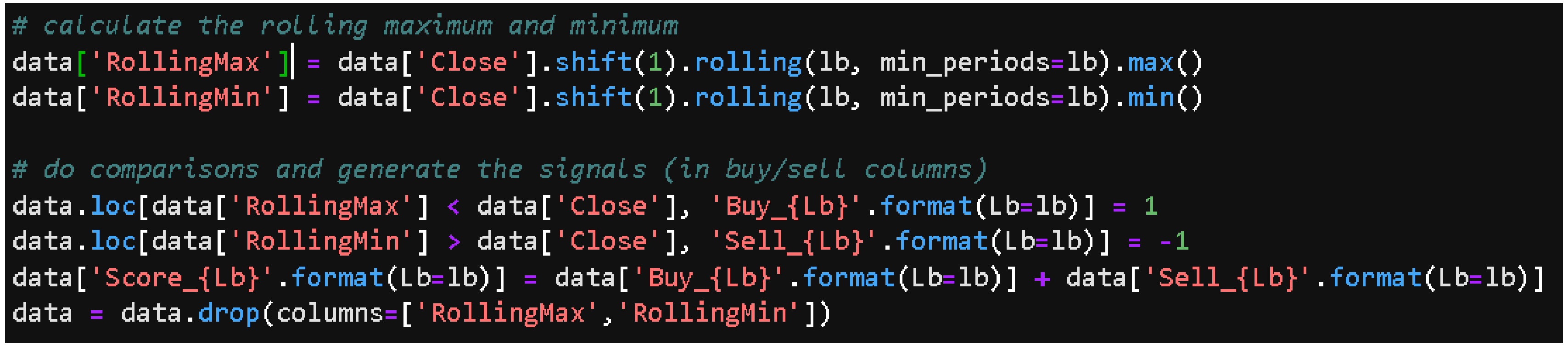
$$\Theta_{i_j t} = \begin{cases} 1 & P_t > \max_{k \in \{1,2,\ldots,i_j\}} P_{t-k} & \textit{Buy Signal} \\ 0 & \min_{k \in \{1,2,\ldots,i_j\}} P_{t-k} < P_t < \max_{k \in \{1,2,\ldots,i_j\}} P_{t-k} & \textit{No Signal} \\ -1 & P_t < \min_{k \in \{1,2,\ldots,i_j\}} P_{t-k} & \textit{Sell Signal} \end{cases}$$

where P_t is the price of the asset (Bitcoin) at time t, and

i_j is the period of look-back, for example, 60 (days)

TREND-FOLLOWING STRATEGY - SUPPORT & RESISTENCE

We can see that a single lookback signal is similar to an indication of whether the current price is breaking an "n-day resistance" or "n-day support". In our model, we believe that if we receive an "n-day lookback buy signal" (current close price greater than the n-day rolling maximum), then the price is breaking an "n-day resistance" and thus the price of Bitcoin is going to rise in the future. In this case, the investor should buy the asset, i.e. Bitcoin, and vice versa for n-day rolling minimum as "n-day support".



```
# calculate the rolling maximum and minimum
data['RollingMax'] = data['Close'].shift(1).rolling(lb, min_periods=lb).max()
data['RollingMin'] = data['Close'].shift(1).rolling(lb, min_periods=lb).min()

# do comparisons and generate the signals (in buy/sell columns)
data.loc[data['RollingMax'] < data['Close'], 'Buy_{Lb}'.format(Lb=lb)] = 1
data.loc[data['RollingMin'] > data['Close'], 'Sell_{Lb}'.format(Lb=lb)] = -1
data['Score_{Lb}'.format(Lb=lb)] = data['Buy_{Lb}'.format(Lb=lb)] + data['Sell_{Lb}'.format(Lb=lb)]
data = data.drop(columns=['RollingMax','RollingMin'])
```

TREND-FOLLOWING STRATEGY - WEIGHTED SIGNALS

For the weighted average of the lookback signals that we focus in our model, the score of it can be mathematically expressed as:

$$\widetilde{\Theta}_t = \vec{w} \cdot \vec{\Theta}_t = w_{i_1}\Theta_{i_1 t} + w_{i_2}\Theta_{i_2 t} + \cdots + w_{i_N}\Theta_{i_N t} \text{ such that } \textstyle\sum_{j=1}^{N} w_{i_j} = 1$$

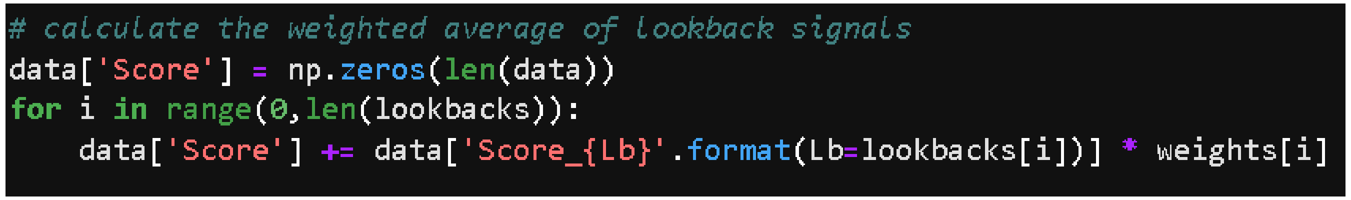
where $\widetilde{\Theta}_t$ is the weighted score at time t, N is the number of different look-backs,

$\vec{w} = (w_{i_1}\ w_{i_2}\ \ldots\ w_{i_N})^T$ are the weights of the look-back signals at time t, and

$\vec{\Theta}_t = (\Theta_{i_1 t}\ \Theta_{i_2 t}\ \ldots\ \Theta_{i_N t})^T$ are the look-back signals (-1, 0, 1) at time t.

The buy/sell signal Z at time t is determined by:

$$Z_t = \begin{cases} 1 & \widetilde{\Theta}_t \geq \tau_B & \textit{Buy Signal} \\ 0 & \tau_s \leq \widetilde{\Theta}_t \leq \tau_B & \textit{No Signal} \\ -1 & \widetilde{\Theta}_t \leq \tau_S & \textit{Sell Signal} \end{cases}$$



```python
# calculate the weighted average of lookback signals
data['Score'] = np.zeros(len(data))
for i in range(0,len(lookbacks)):
    data['Score'] += data['Score_{Lb}'.format(Lb=lookbacks[i])] * weights[i]
```

where τ_B and τ_S are the thresholds for buy and sell signals respectively

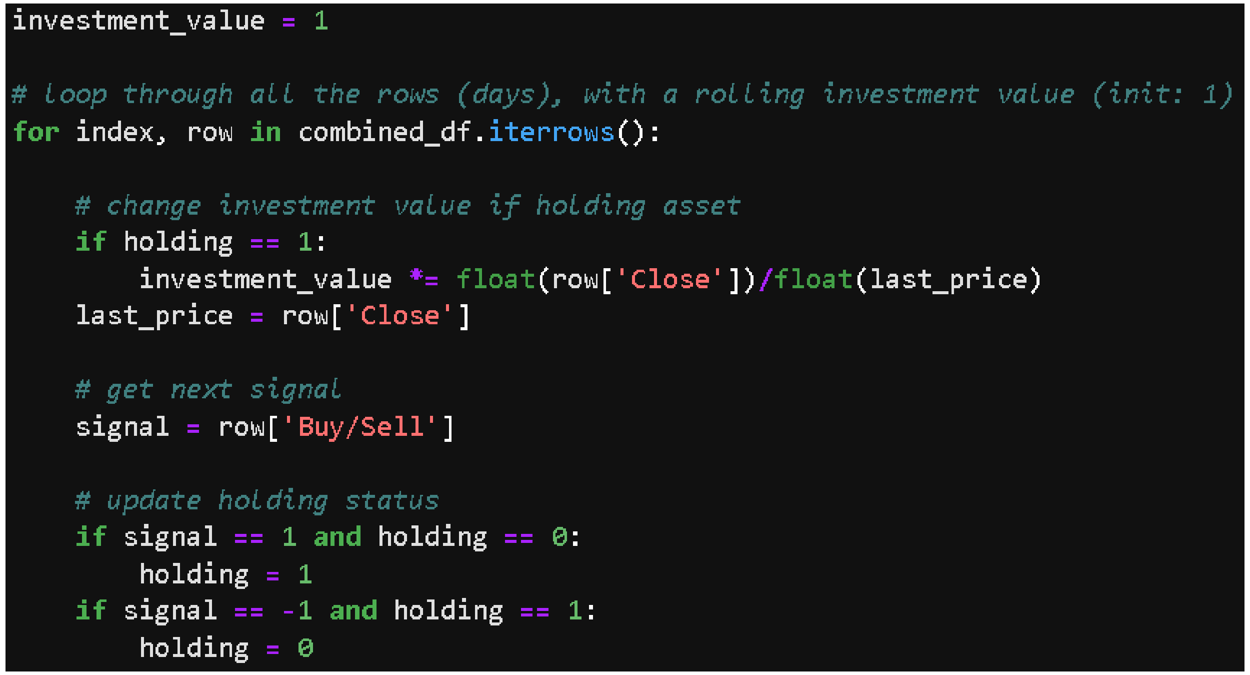
HELPER FUNCTION REVISITED (RETURN_CALCULATION.PY)

We further assume that there is no transaction cost, and the interest rate effect is negligible. So the investment value changes only when the investor invests all his/her capital in Bitcoin, i.e. when the latest signal is a Buy Signal. Also, the investor holds the asset until there is a Sell Signal. Mathematically, the investment value ($1<t\leq T$) has the following property:

$$I_t = \begin{cases} I_{t-1} & Z_{t-1} = -1;\ \text{or}\ Z_{t-1} = 0 \text{ and investor is not holding the asset} \\ I_{t-1} \cdot \dfrac{P_t}{P_{t-1}} & Z_{t-1} = 1,;\ \text{or}\ Z_{t-1} = 0 \text{ and investor is holding the asset} \end{cases}$$

where T is the last day of the investment period,

i.e. the last day of training / testing period.

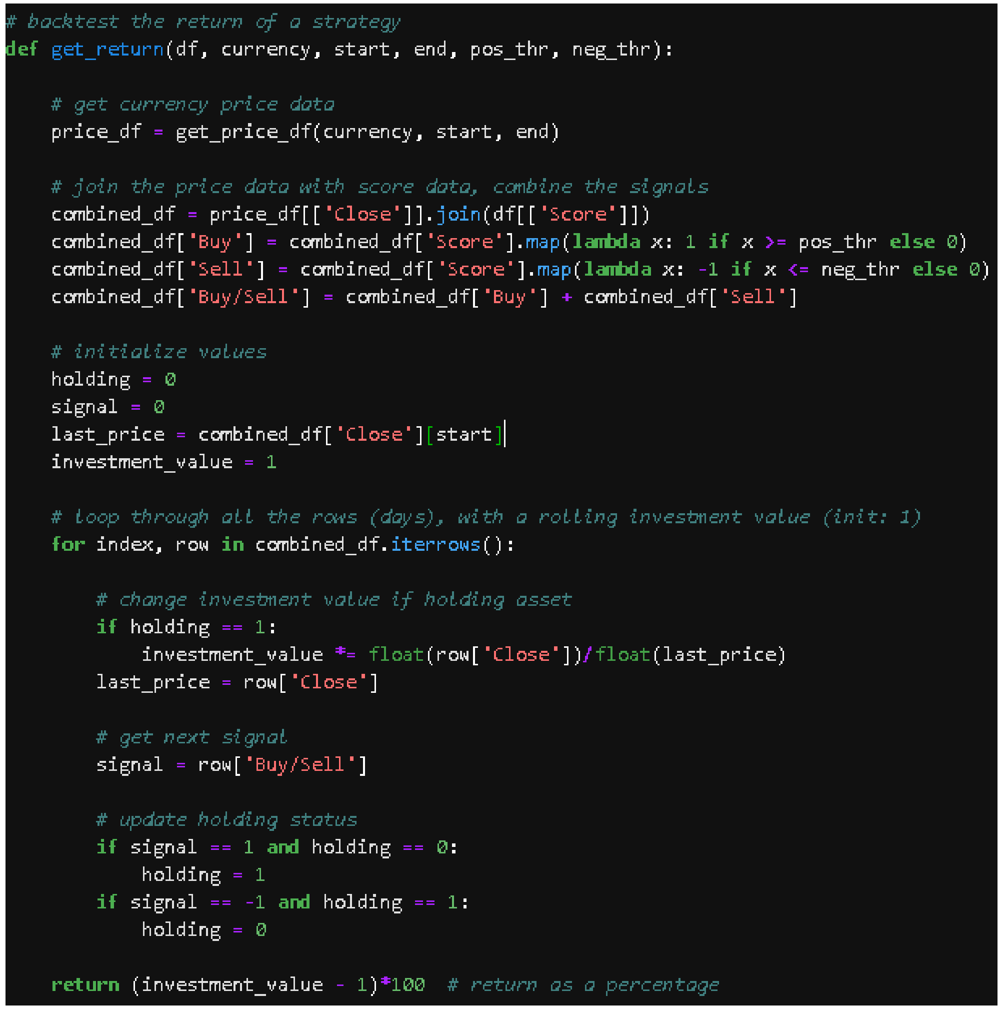


```
investment_value = 1

# loop through all the rows (days), with a rolling investment value (init: 1)
for index, row in combined_df.iterrows():

    # change investment value if holding asset
    if holding == 1:
        investment_value *= float(row['Close'])/float(last_price)
    last_price = row['Close']

    # get next signal
    signal = row['Buy/Sell']

    # update holding status
    if signal == 1 and holding == 0:
        holding = 1
    if signal == -1 and holding == 1:
        holding = 0
```

HELPER FUNCTION REVISITED (RETURN_CALCULATION.PY)

get_return() function transforms scores and thresholds in a period into the investment return of that period

In this function, we will first create a column of buy/sell signals using the given scores. Then, it will loop through all the inner-join rows (price join score), and go into a for loop: Calculates the daily investment value one by one, finally reach the last day of the investment period, on which the investment value can deduce the total investment return.



```python
# backtest the return of a strategy
def get_return(df, currency, start, end, pos_thr, neg_thr):

    # get currency price data
    price_df = get_price_df(currency, start, end)

    # join the price data with score data, combine the signals
    combined_df = price_df[['Close']].join(df[['Score']])
    combined_df['Buy'] = combined_df['Score'].map(lambda x: 1 if x >= pos_thr else 0)
    combined_df['Sell'] = combined_df['Score'].map(lambda x: -1 if x <= neg_thr else 0)
    combined_df['Buy/Sell'] = combined_df['Buy'] + combined_df['Sell']

    # initialize values
    holding = 0
    signal = 0
    last_price = combined_df['Close'][start]
    investment_value = 1

    # loop through all the rows (days), with a rolling investment value (init: 1)
    for index, row in combined_df.iterrows():

        # change investment value if holding asset
        if holding == 1:
            investment_value *= float(row['Close'])/float(last_price)
        last_price = row['Close']

        # get next signal
        signal = row['Buy/Sell']

        # update holding status
        if signal == 1 and holding == 0:
            holding = 1
        if signal == -1 and holding == 1:
            holding = 0

    return (investment_value - 1)*100  # return as a percentage
```

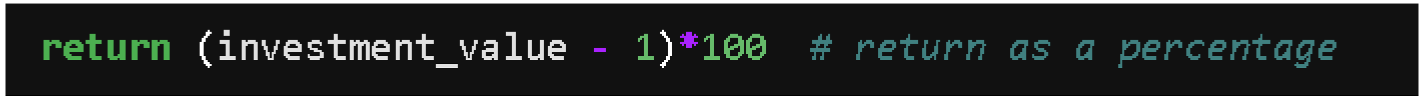
HELPER FUNCTION REVISITED (RETURN_CALCULATION.PY)

In our strategy, we will assume that the investor has 1 dollar initially ($I_1 = 1$), and we can calculate the total return in percentage.

I_T can be obtained by iterating the property of I_t through the whole training / testing period.

The total trading return Π is given by:

$$\Pi(\vec{w}, \tau_B, \tau_s) = \frac{I_T - 1}{1} \cdot 100\% = (I_T - 1) \cdot 100\%$$



```
return (investment_value - 1)*100  # return as a percentage
```

TREND-FOLLOWING STRATEGY - TOTAL RETURN

I_T can be obtained by iterating the property of I_t through the whole training / testing period.

The total trading return Π is given by:

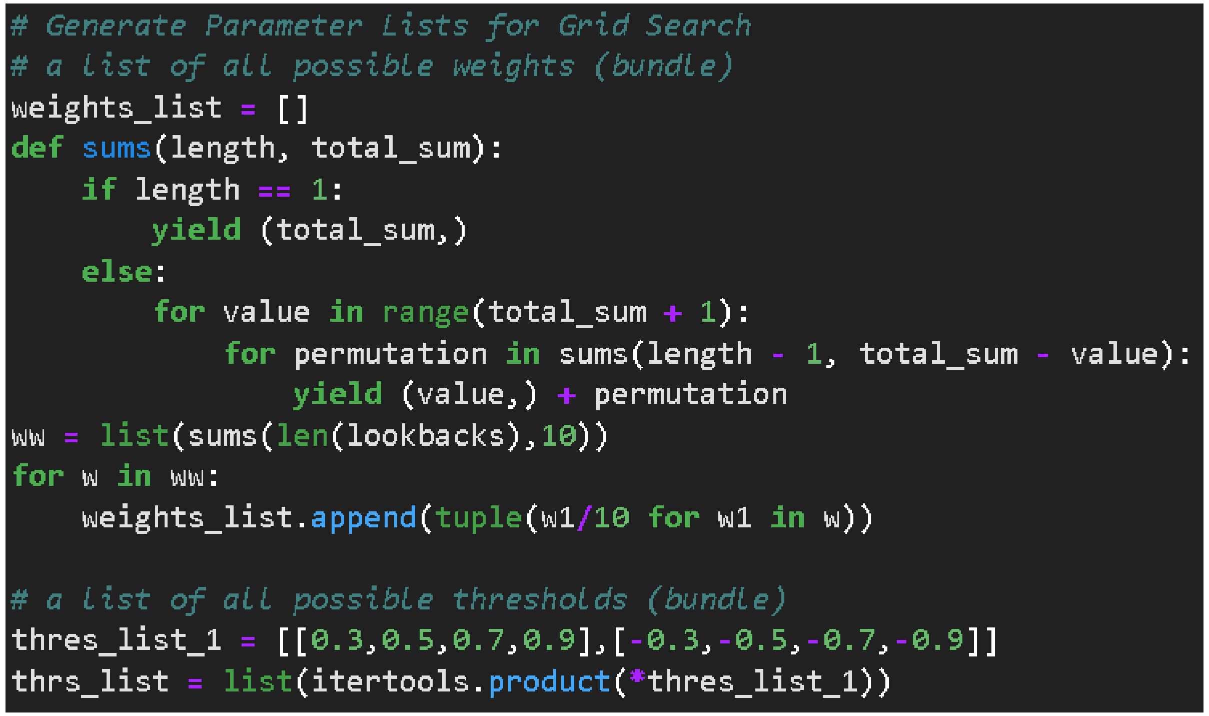
$$\Pi(\vec{w}, \tau_B, \tau_s) = \frac{I_T - 1}{1} \cdot 100\% = (I_T - 1) \cdot 100\%$$

Mathematically, we search for $\mathrm{argmax}_{\vec{w}, \tau_B, \tau_s} \Pi$, where Π is the trading return (in %). For simplicity, we use a grid search to find the best parameters, i.e. we iterate through all the possible values of the parameters and choose the onces with the highest trading return. The lookbacks determining $\vec{\Theta}_t$ are predetermined as 20, 30, 60 and 90 days in our model for simplicity (to avoid super-long runtime in training), i.e. we adopt $N = 4$, and $(i_1, i_2, i_3) = (20,30,60,90)$. For parameters $\vec{w} = (w_{i_1}\ w_{i_2} \ldots w_{i_N})^T$ and $\{\tau_B, \tau_s\} \in [-1,1]$, we assume:

$$\vec{w} \in \{0.1, 0.2, \ldots, 1.0\}^4 \in \mathbb{R}^4 \text{ and}$$

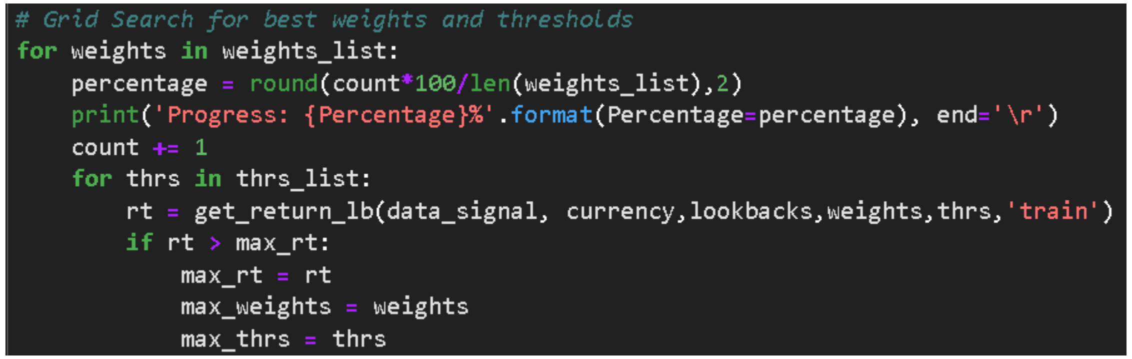
$$(\tau_B, \tau_S) \in \{0.3, 0.5, 0.7, 0.9\} \times \{-0.3, -0.5, -0.7, -0.9\}$$

to yield the greatest trading return on the backtest of training data using grid search.



```python
# Generate Parameter Lists for Grid Search
# a list of all possible weights (bundle)
weights_list = []
def sums(length, total_sum):
    if length == 1:
        yield (total_sum,)
    else:
        for value in range(total_sum + 1):
            for permutation in sums(length - 1, total_sum - value):
                yield (value,) + permutation
ww = list(sums(len(lookbacks),10))
for w in ww:
    weights_list.append(tuple(w1/10 for w1 in w))

# a list of all possible thresholds (bundle)
thres_list_1 = [[0.3,0.5,0.7,0.9],[-0.3,-0.5,-0.7,-0.9]]
thrs_list = list(itertools.product(*thres_list_1))
```

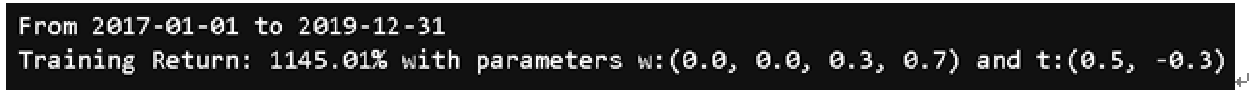
TREND-FOLLOWING STRATEGY (WITH HELPER FUNCTION)



```python
# Grid Search for best weights and thresholds
for weights in weights_list:
    percentage = round(count*100/len(weights_list),2)
    print('Progress: {Percentage}%'.format(Percentage=percentage), end='\r')
    count += 1
    for thrs in thrs_list:
        rt = get_return_lb(data_signal, currency,lookbacks,weights,thrs,'train')
        if rt > max_rt:
            max_rt = rt
            max_weights = weights
            max_thrs = thrs
```

Regarding the result of the grid search, the training result indicates:

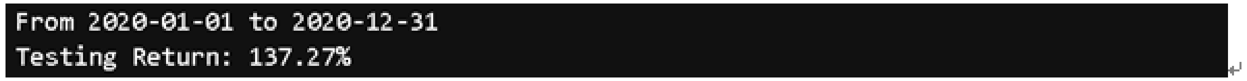
- 1145.01% return in 3 years with $\vec{w} = (0.0,0.0,0.3,0.7)^T$ and $(\tau_B, \tau_S) = (0.5, -0.3)$, which is equivalent to 131.77% return p.a.



```
From 2017-01-01 to 2019-12-31
Training Return: 1145.01% with parameters w:(0.0, 0.0, 0.3, 0.7) and t:(0.5, -0.3)
```

The testing result using $\vec{w} = (0.0,0.0,0.3,0.7)^T$ and $(\tau_B, \tau_S) = (0.5, -0.3)$, indicates:

- 137.27% return in one year, which is not far from the training return 131.77% (p.a.).



```
From 2020-01-01 to 2020-12-31
Testing Return: 137.27%
```

We conclude that the over-fitting in this model is not severe, and the strategy is working well and producing a considerable return.

TREND-FOLLOWING STRATEGY - INSIGHTS

We also found that longer lookback signals tend to be more insightful, as we see the weight on the longest signal is the highest after grid search. Trading with the weights and thresholds chosen by the grid search in the current result is indeed equivalent to "buy when 60-day signal says buy; and sell when 90-day signals says sell". The grid search can be further improved in the future to include more possible values, but the runtime to train the model will be much longer in that case. For the signals in our model, we do not use even longer signals, as (i) the training and testing periods are short, (ii) Bitcoin has only been available for a few years (so we cannot have more data), and (iii) the general behavior of Bitcoin we want to examine may only be happening after 2017. Before 2017, Bitcoin has a comparatively very low price, possibly because of the lack of awareness and acceptance.

TREND-FOLLOWING STRATEGY - PLOTS

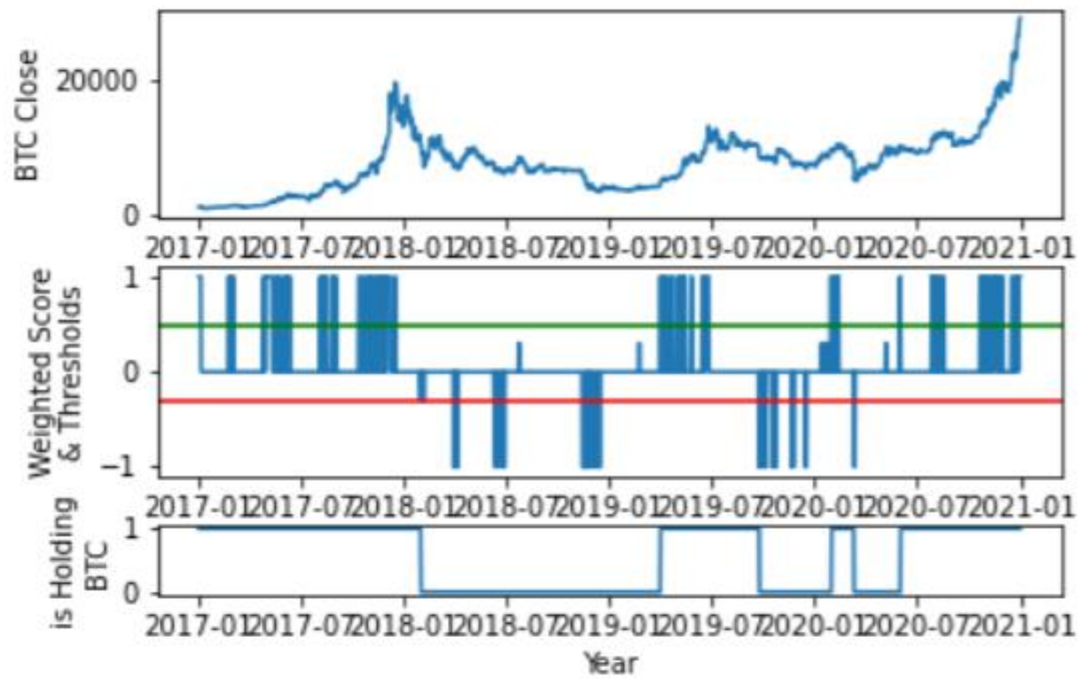


Figure Results: Training & Testing Periods

Training Return (3 Years): 1145.01%
Testing Return (1 Year): 137.27%

Strategy is working well and produces a considerable return.

TREND-FOLLOWING STRATEGY - PLOTS

Extend the model to earlier periods

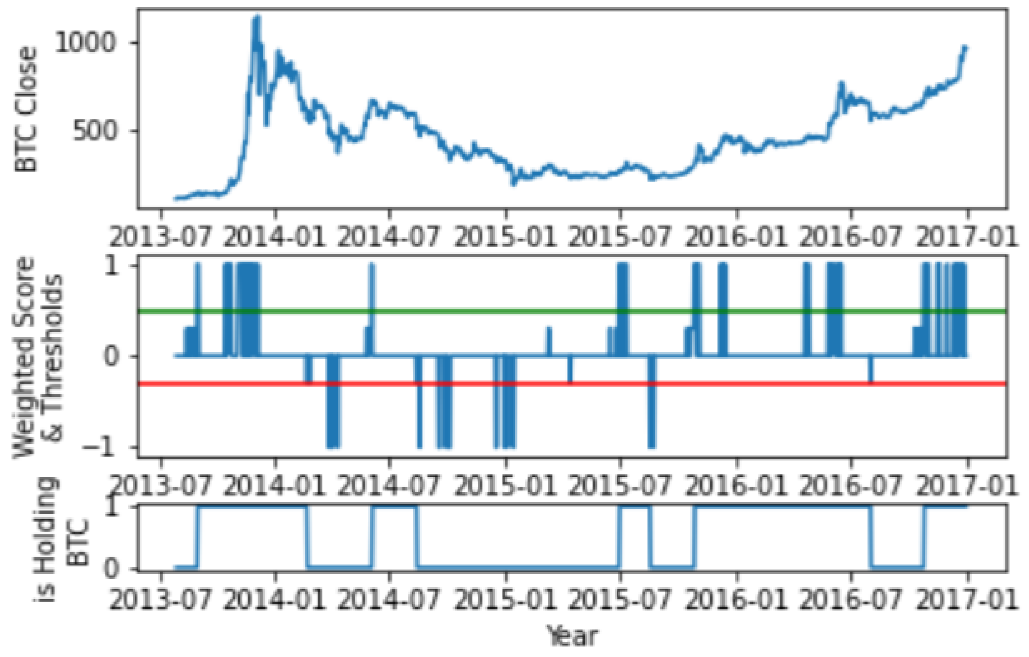


Figure *Results: Before 2017 (before the training & testing periods)*

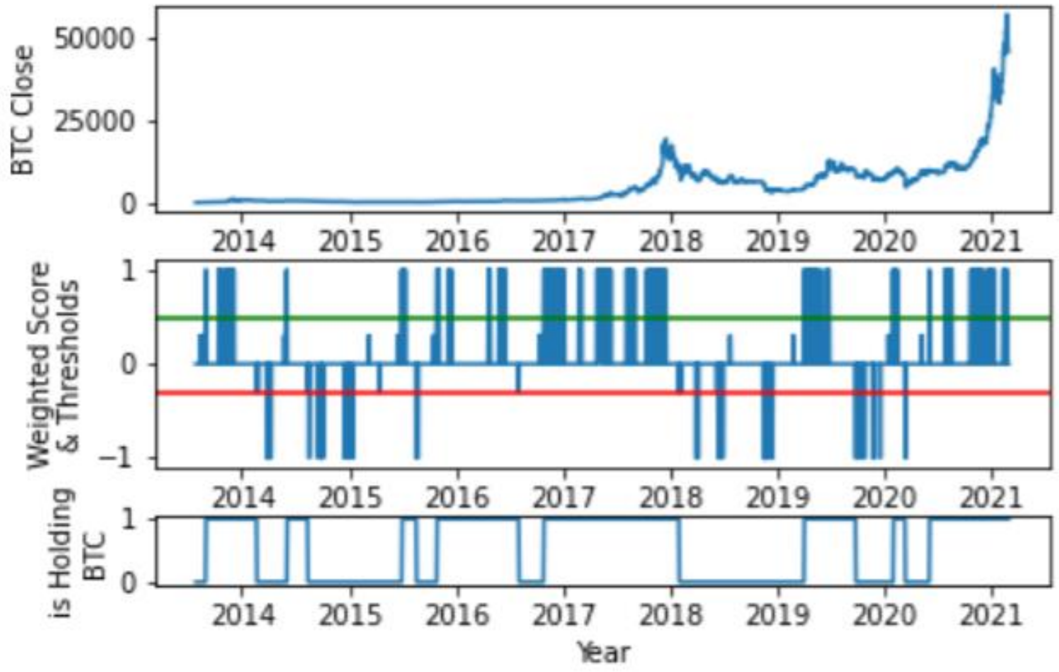


Figure *Results: All Closes staring from 2013*

SENTIMENT ANALYSIS - MODELS

For full codes of this model (Part III), please refer to these files:

part_3a_sentiment_data_preprocessing.ipynb

part_3b_sentiment_analysis return_calculation.py

We have researched for different methods including VADER and TextBlob for evaluating the sentiment scores, and finally we chose VADER to be included in our model because it is attuned specifically to sentiments expressed in social media (e.g. Twitter) and is able to give more detailed information (e.g., 'positive', 'negative', 'compound', 'neutral'). VADER uses a list of lexical features (e.g. word) which are labeled as positive or negative according to their semantic orientation to calculate the text sentiment and returns the probability of a given input sentence to be positive, negative, and neutral.

SENTIMENT ANALYSIS - SENTIMENT SCORES

The first step is to obtain the sentiment score of the tweets, which can be done by the following:

```python
import pandas as pd
import numpy as np
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
```

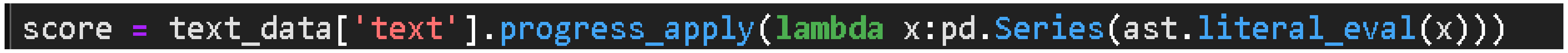
```python
text_data_score = text_data['text'].progress_apply(lambda Description: sid.polarity_scores(Description))
```

Output:

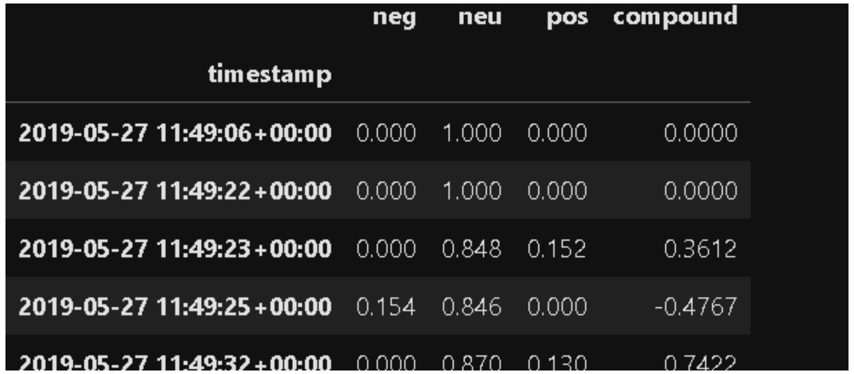
```
timestamp
2019-05-27 11:49:06+00:00    {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...
2019-05-27 11:49:22+00:00    {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...
2019-05-27 11:49:23+00:00    {'neg': 0.0, 'neu': 0.848, 'pos': 0.152, 'comp...
2019-05-27 11:49:25+00:00    {'neg': 0.154, 'neu': 0.846, 'pos': 0.0, 'comp...
```

SENTIMENT ANALYSIS – TAKING AVERAGE

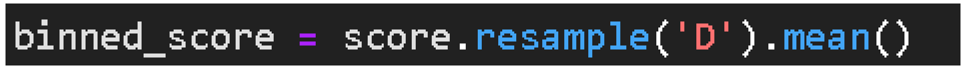
After we obtain the sentiment score of each tweet, we then turn it into data frame, and calculate the daily average sentiment score by taking the daily-mean of the scores of all the tweets on each day using the following codes:



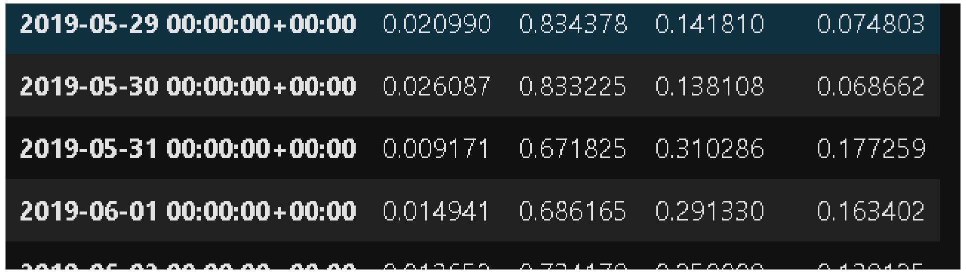
```
score = text_data['text'].progress_apply(lambda x:pd.Series(ast.literal_eval(x)))
```



```
                             neg    neu    pos   compound
                timestamp
2019-05-27 11:49:06+00:00  0.000  1.000  0.000     0.0000
2019-05-27 11:49:22+00:00  0.000  1.000  0.000     0.0000
2019-05-27 11:49:23+00:00  0.000  0.848  0.152     0.3612
2019-05-27 11:49:25+00:00  0.154  0.846  0.000    -0.4767
2019-05-27 11:49:32+00:00  0.000  0.870  0.130     0.7422
```



```
binned_score = score.resample('D').mean()
```



```
2019-05-29 00:00:00+00:00  0.020990  0.834378  0.141810  0.074803
2019-05-30 00:00:00+00:00  0.026087  0.833225  0.138108  0.068662
2019-05-31 00:00:00+00:00  0.009171  0.671825  0.310286  0.177259
2019-06-01 00:00:00+00:00  0.014941  0.686165  0.291330  0.163402
[illegible]
```

SENTIMENT ANALYSIS - EXPLAINED

After lots of preprocessing (which we skipped here), the data is then transformed to be an input of the following model. We then train the Long short-term memory (LSTM) model with the daily sentiment score (calculated in the previous step) and price change history (from the dataset) in the past 30 days to predict the price change in the next day. LSTM is an artificial recurrent neural network (RNN) architecture with feedback connections, which is especially good at making predictions based on time series data even with varying lags between input samples. It could also deal with the vanishing gradient problem that could happen when training traditional RNNs. The data input is normalized to minimize the impact of extreme values on the model. Our model uses the mean square error as a loss function and Adam as the optimization method. We performed grid-search with cross-validation to fine-tune the best parameter, which we found is 256 batch size and 200 epochs.

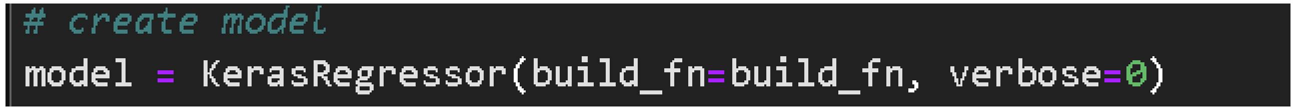
SENTIMENT ANALYSIS - MODEL

Building of the Model...

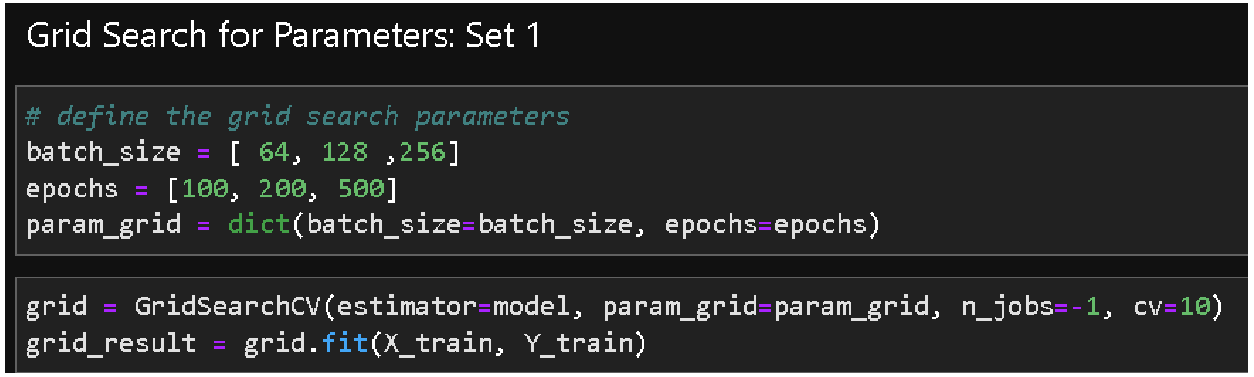
```python
def buildManyToOneModelFn(shape):
    def buildManyToOneModel():
        model = Sequential()
        model.add(LSTM(10, input_length=shape[1], input_dim=shape[2]))
        # output shape: (1, 1)
        model.add(Dense(1))
        model.compile(loss="mse", optimizer="adam")
        model.summary()
        return model
    return buildManyToOneModel
```

SENTIMENT ANALYSIS - TRAINING THE MODEL

Training the Model… Including grid-search for different sets of parameters



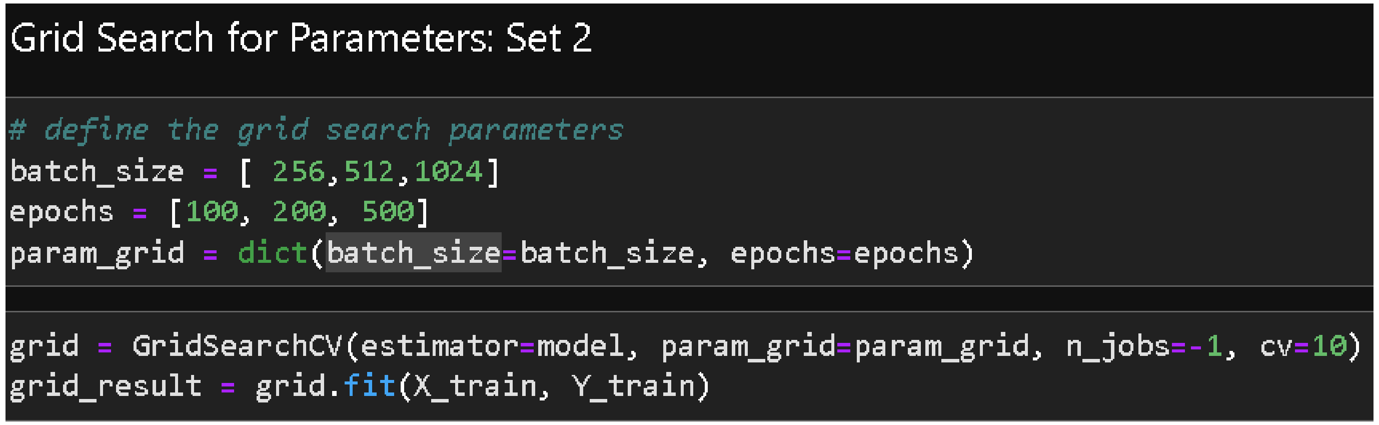
```
# create model
model = KerasRegressor(build_fn=build_fn, verbose=0)
```



Grid Search for Parameters: Set 1

```
# define the grid search parameters
batch_size = [ 64, 128 ,256]
epochs = [100, 200, 500]
param_grid = dict(batch_size=batch_size, epochs=epochs)
```

```
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=10)
grid_result = grid.fit(X_train, Y_train)
```



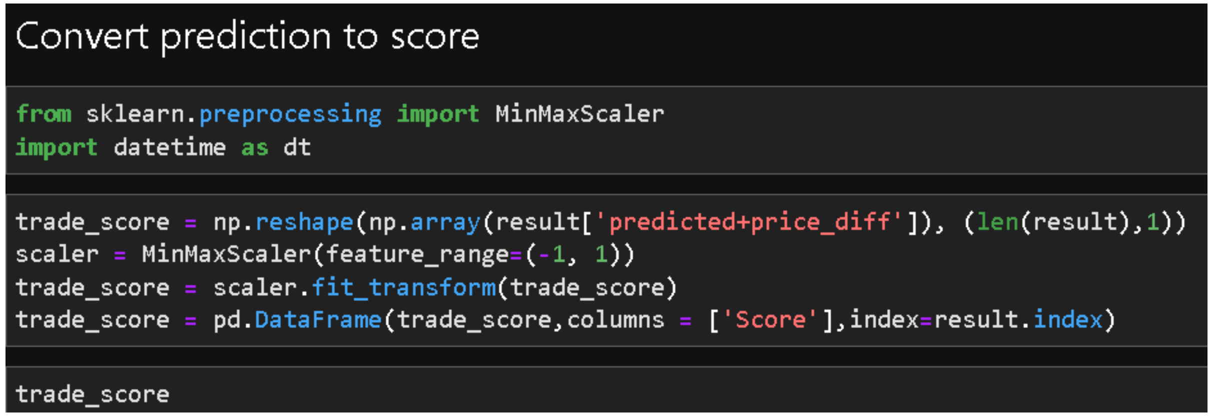
Grid Search for Parameters: Set 2

```
# define the grid search parameters
batch_size = [ 256,512,1024]
epochs = [100, 200, 500]
param_grid = dict(batch_size=batch_size, epochs=epochs)
```

```
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=10)
grid_result = grid.fit(X_train, Y_train)
```

SENTIMENT ANALYSIS - SCORES

Finally, we normalize the predicted price change into the range between -1 and 1. If the score is closer to 1 on a particular day, it indicates that buying the asset (Bitcoin) has a higher probability of getting a higher positive return.

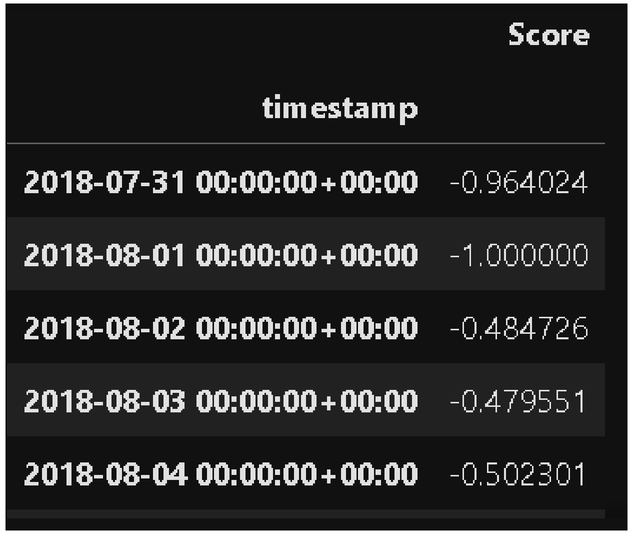


Convert prediction to score

```
from sklearn.preprocessing import MinMaxScaler
import datetime as dt
```

```
trade_score = np.reshape(np.array(result['predicted+price_diff']), (len(result),1))
scaler = MinMaxScaler(feature_range=(-1, 1))
trade_score = scaler.fit_transform(trade_score)
trade_score = pd.DataFrame(trade_score,columns = ['Score'],index=result.index)
```

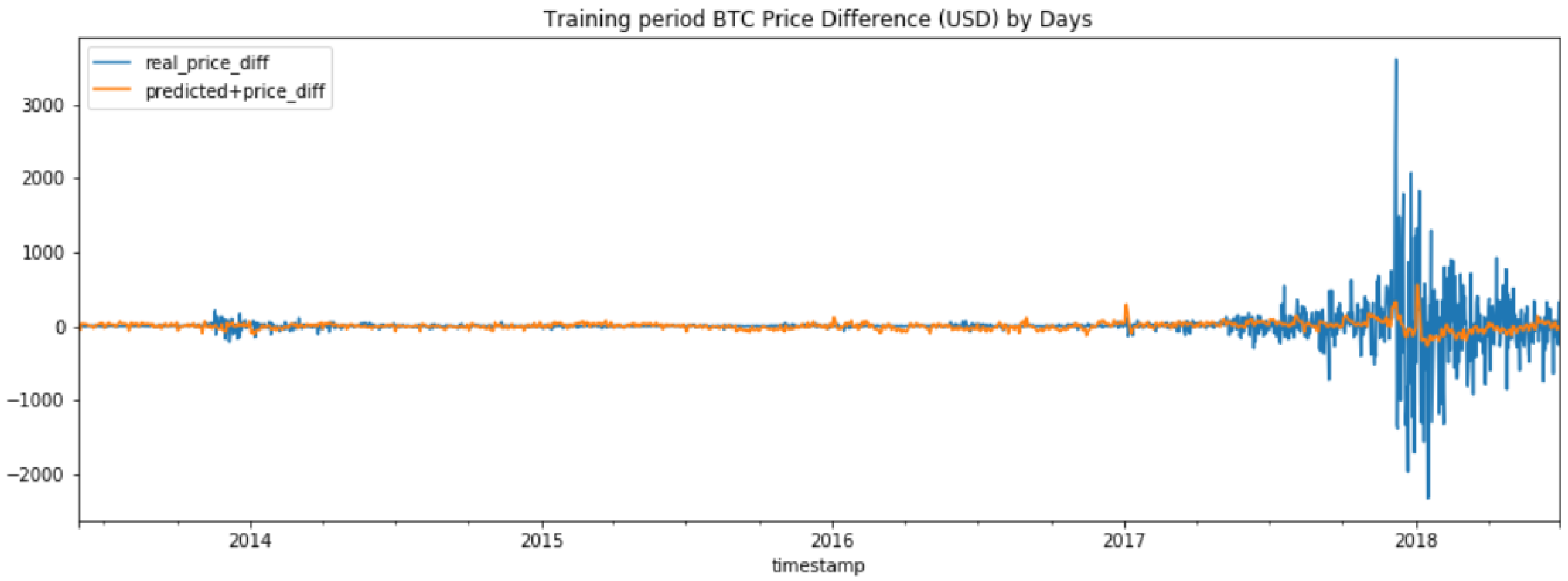
```
trade_score
```



	Score
timestamp	
2018-07-31 00:00:00+00:00	-0.964024
2018-08-01 00:00:00+00:00	-1.000000
2018-08-02 00:00:00+00:00	-0.484726
2018-08-03 00:00:00+00:00	-0.479551
2018-08-04 00:00:00+00:00	-0.502301

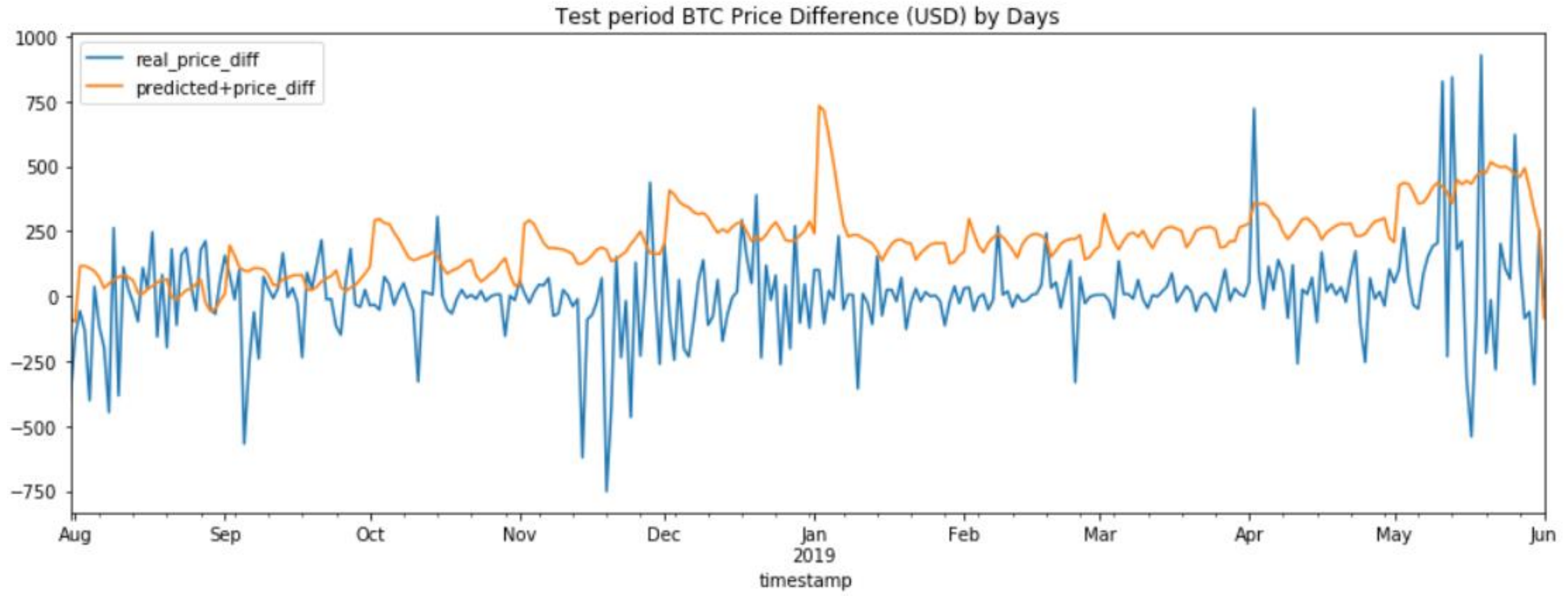
SENTIMENT ANALYSIS - PLOTS

Train Period:



SENTIMENT ANALYSIS - PLOTS

Test Period:



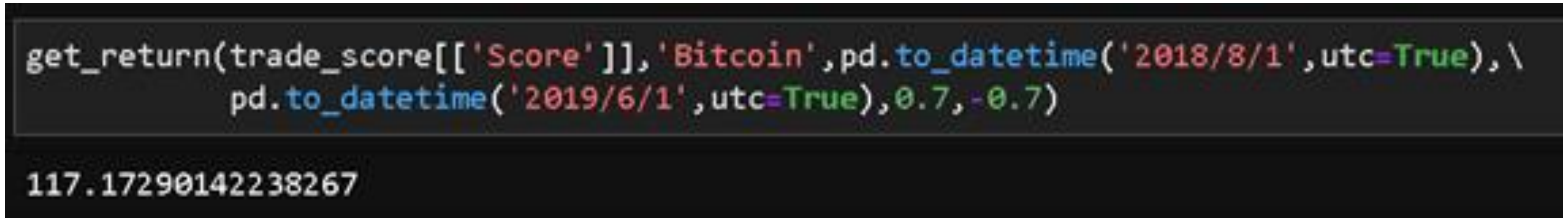
The model is not that accurate, need further tuned in the future OR simply tweets is not a determining factor!

RESULT SUMMARY - INSIGHTS

As we observe from the figures in the previous slides, we observe that the prediction in price difference is not very accurate and tend to be a bit higher than actual. Thus, the model has to be further tuned in the future, or that tweets is not a determining factor of the price of Bitcoin

SENTIMENT ANALYSIS - CALCULATE RETURN (HELPER FUNCTION)

The result of this model is as follows:



```
get_return(trade_score[['Score']],'Bitcoin',pd.to_datetime('2018/8/1',utc=True),\
          pd.to_datetime('2019/6/1',utc=True),0.7,-0.7)

117.17290142238267
```

using the threshold as (0.7,-0.7). The return (117% in the period 01-08-2018 to 01-06-2919) is comparatively low to other models.

We will see if the sentiment factor is useful when attempting to combine different factors in the combined model in the next part.

SCORE-COMBINED MODEL - EXPLAINED

For full codes of this model (Part IV), please refer to these files:
part_4_scores_combined.ipynb return_calculation.py

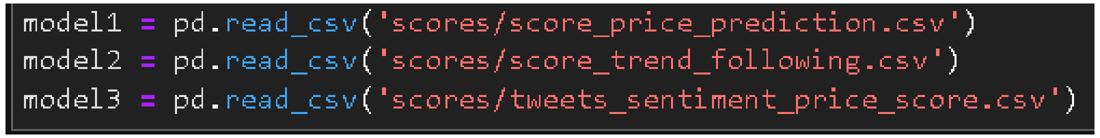
We use linear combination of scores with assigned weights using argmax to maximize the return. It can be expressed in follows:

$$\vec{s}_w = a\vec{s}_1 + b\vec{s}_2 + c\vec{s}_w \quad where \quad a + b + c = 1$$

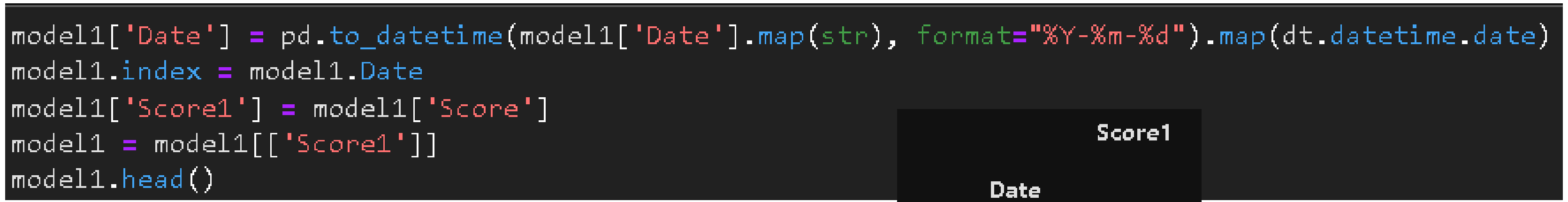
Here, Sw is the weighted score, S1 to S3 is the score produced in Parts 1-3 respectively.
i.e. 1: Price Prediction (with weight a), 2: Trend-Following (with weight b), 3: Sentiment (with weight c)

SCORE-COMBINED MODEL - CODES

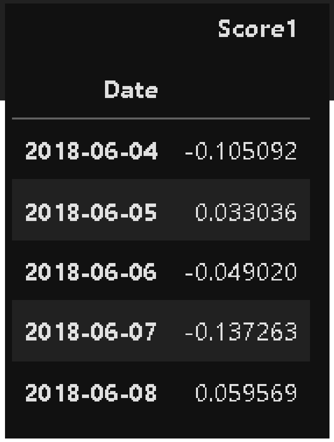
In our codes, we first attempt to merge all the scores: read the score data, use date as the index, and join them together.



```
model1 = pd.read_csv('scores/score_price_prediction.csv')
model2 = pd.read_csv('scores/score_trend_following.csv')
model3 = pd.read_csv('scores/tweets_sentiment_price_score.csv')
```

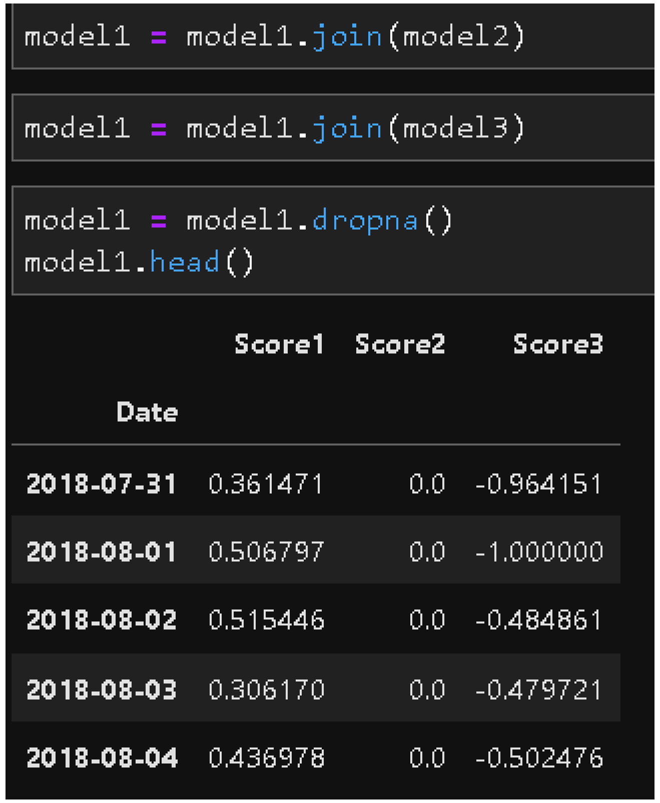


```
model1['Date'] = pd.to_datetime(model1['Date'].map(str), format="%Y-%m-%d").map(dt.datetime.date)
model1.index = model1.Date
model1['Score1'] = model1['Score']
model1 = model1[['Score1']]
model1.head()
```



```
              Score1
Date
2018-06-04 -0.105092
2018-06-05  0.033036
2018-06-06 -0.049020
2018-06-07 -0.137263
2018-06-08  0.059569
```

---- >>



```
model1 = model1.join(model2)
```

```
model1 = model1.join(model3)
```

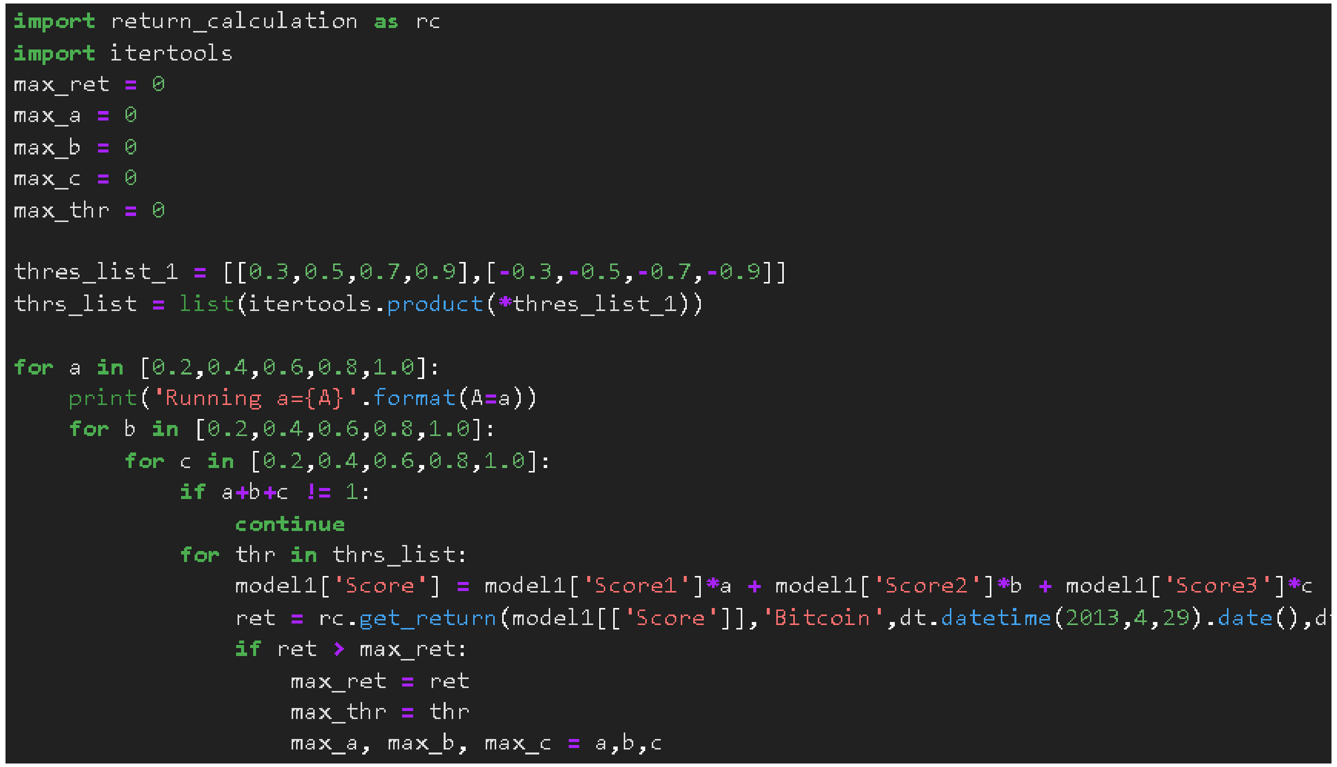
```
model1 = model1.dropna()
model1.head()
```

```
              Score1  Score2     Score3
Date
2018-07-31  0.361471     0.0  -0.964151
2018-08-01  0.506797     0.0  -1.000000
2018-08-02  0.515446     0.0  -0.484861
2018-08-03  0.306170     0.0  -0.479721
2018-08-04  0.436978     0.0  -0.502476
```

SCORE-COMBINED MODEL - GRID SEARCH

The codes we used to grid-search for the best a, b, c and threshold-pair.

We want to seek the best parameter combination producing the highest return (with helper function).



```python
import return_calculation as rc
import itertools
max_ret = 0
max_a = 0
max_b = 0
max_c = 0
max_thr = 0

thres_list_1 = [[0.3,0.5,0.7,0.9],[-0.3,-0.5,-0.7,-0.9]]
thrs_list = list(itertools.product(*thres_list_1))

for a in [0.2,0.4,0.6,0.8,1.0]:
    print('Running a={A}'.format(A=a))
    for b in [0.2,0.4,0.6,0.8,1.0]:
        for c in [0.2,0.4,0.6,0.8,1.0]:
            if a+b+c != 1:
                continue
            for thr in thrs_list:
                model1['Score'] = model1['Score1']*a + model1['Score2']*b + model1['Score3']*c
                ret = rc.get_return(model1[['Score']],'Bitcoin',dt.datetime(2013,4,29).date(),d
                if ret > max_ret:
                    max_ret = ret
                    max_thr = thr
                    max_a, max_b, max_c = a,b,c
```

We assume:

$$a \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$$

$$b \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$$

$$c \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$$

$$(\tau_B, \tau_S) \in \{0.3, 0.5, 0.7, 0.9\} \times \{-0.3, -0.5, -0.7, -0.9\}$$

$$\text{where } a + b + c = 1$$

PREDICTION MODEL - RESULTS

We assume:

$$a \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$$

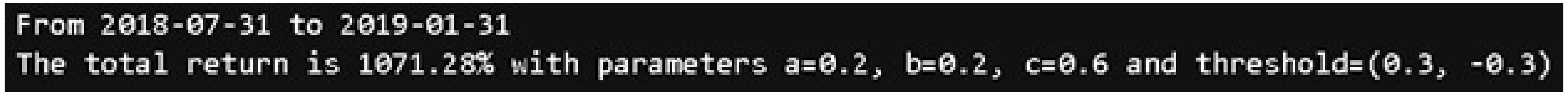
$$b \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$$

$$c \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$$

$$(\tau_B, \tau_S) \in \{0.3, 0.5, 0.7, 0.9\} \times \{-0.3, -0.5, -0.7, -0.9\}$$

$$\text{where } a + b + c = 1$$

The best parameter is a = 0.2, b = 0.2, c = 0.6



```
From 2018-07-31 to 2019-01-31
The total return is 1071.28% with parameters a=0.2, b=0.2, c=0.6 and threshold=(0.3, -0.3)
```

The resulting return of 1071.28% (185 days), which is equivalent to 12736% p.a., is better than any individual models. We can also see that model 3 (sentiment) has the highest weight of 0.6

CONCLUSIONS

The three separated models are all proven to be profit-generating using the backtests, as the percentage printed out in the codes. We adopted different innovations and creative ideas, including: combining different models, taking weighted averages with grid-searched weights, using different kinds of data in the same model (numbers and texts), apply stock trading strategies in a cryptocurrency, ... (and more)

The combined model is found to be generating a return even higher than all these three models. We conclude that price, trend and sentiment are useful factors to be considered when trading Bitcoin; and combining factor can enhance the trading strategy in this case. Sentiment is also a factor that works much better when combing with other factors, and in that case, sentiment has a high weight and is important!

THANK YOU