Relationship with cryptocurrency through Twitter sentiment analysis

17102040 Tae Hyeon Kwon
17102042 Min Seon Kim
17102049 Jae Min Park
17102058 Chang Wan Woo

Contents

1. Background

Background

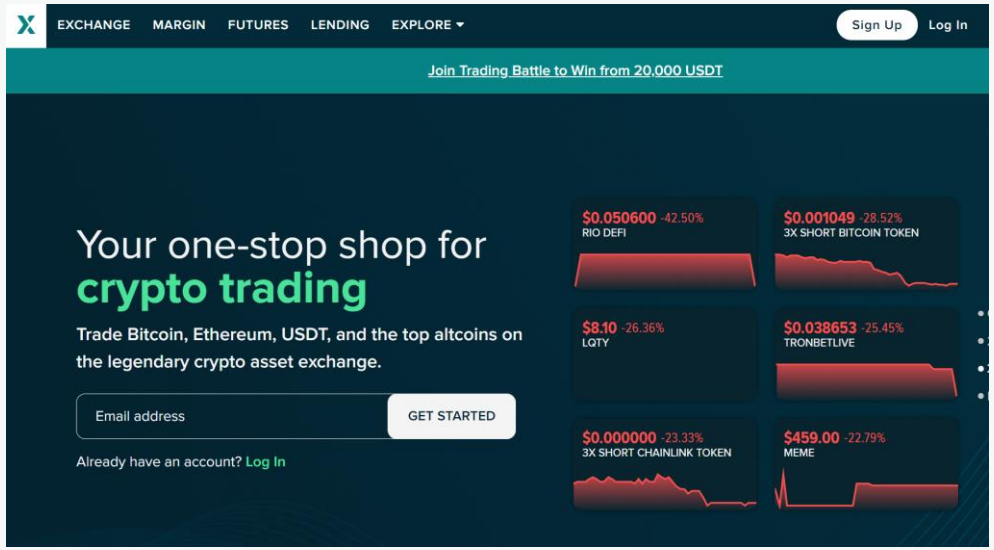
Sentimental Analysis

We analyzed the correlation between the results of tweets through sentiment analysis and the price of Bitcoin.

Basic Idea - Datasets

1. Twitter data that includes keywords related to Cryptocurrency(Bitcoin, Ethereum)
2. Cryptocurrency real time data from poloniex

Basic Idea - Datasets



{"date":1623240000,"high":0.07316335,"low":0.06986294,"open":0.07255866,"close":0.07059776,"volume":187.13904366,"quoteVolume":2614.5058012,"weightedAverage":0.07157721},
{"date":1623254400,"high":0.07156061,"low":0.07059776,"open":0.07059776,"close":0.07062812,"volume":82.94468636,"quoteVolume":1168.47306904,"weightedAverage":0.07098553},
{"date":1623268800,"high":0.07086619,"low":0.06934391,"open":0.07063597,"close":0.06990099,"volume":16.68207522,"quoteVolume":238.06462686,"weightedAverage":0.07007372},
{"date":1623283200,"high":0.06991604,"low":0.0691,"open":0.06980018,"close":0.06932969,"volume":117.95839908,"quoteVolume":1694.02048566,"weightedAverage":0.06963221},
{"date":1623297600,"high":0.06940254,"low":0.06886819,"open":0.06935706,"close":0.06901,"volume":119.08204015,"quoteVolume":1725.12870836,"weightedAverage":0.06902791},
{"date":1623312000,"high":0.0691469,"low":0.06755658,"open":0.06892611,"close":0.06781457,"volume":418.14836031,"quoteVolume":6115.88235435,"weightedAverage":0.06837089},
{"date":1623326400,"high":0.06820399,"low":0.0670513,"open":0.06783,"close":0.06805385,"volume":98.7985436,"quoteVolume":1455.15358428,"weightedAverage":0.06789561},
{"date":1623340800,"high":0.06807439,"low":0.0670513,"open":0.06807439,"close":0.0670513,"volume":129.25897141,"quoteVolume":1915.53062451,"weightedAverage":0.06747945},
{"date":1623355200,"high":0.06797819,"low":0.06686653,"open":0.0670513,"close":0.06729566,"volume":17.58781387,"quoteVolume":260.52158145,"weightedAverage":0.06751},
{"date":1623369600,"high":0.06742499,"low":0.0666759,"open":0.067278,"close":0.06708076,"volume":116.58548029,"quoteVolume":1741.28212694,"weightedAverage":0.06695381},
{"date":1623384000,"high":0.06715464,"low":0.0658151,"open":0.06705541,"close":0.06583496,"volume":207.16253167,"quoteVolume":3109.95479891,"weightedAverage":0.06661271},
{"date":1623398400,"high":0.06647032,"low":0.06581,"open":0.0658428,"close":0.06611345,"volume":157.03675532,"quoteVolume":2375.77630289,"weightedAverage":0.06609913},
{"date":1623412800,"high":0.0662,"low":0.06550001,"open":0.06609038,"close":0.06565389,"volume":63.66330661,"quoteVolume":965.32282086,"weightedAverage":0.06595027},
{"date":1623427200,"high":0.06573466,"low":0.06438308,"open":0.06556674,"close":0.06438308,"volume":19.33306763,"quoteVolume":297.22443182,"weightedAverage":0.06504535},
{"date":1623441600,"high":0.06444038,"low":0.06274981,"open":0.06437373,"close":0.06309774,"volume":25.22643241,"quoteVolume":397.20872758,"weightedAverage":0.06350926},
{"date":1623456000,"high":0.06481867,"low":0.06281043,"open":0.06310925,"close":0.06468623,"volume":109.83299779,"quoteVolume":1723.0992202,"weightedAverage":0.06374153},
{"date":1623470400,"high":0.0658,"low":0.06461477,"open":0.06469351,"close":0.0658,"volume":49.79058015,"quoteVolume":766.14093395,"weightedAverage":0.0649888},
{"date":1623484800,"high":0.06733563,"low":0.065493,"open":0.06581,"close":0.06709096,"volume":46.17936588,"quoteVolume":694.8811671,"weightedAverage":0.06645649},
{"date":1623499200,"high":0.06814206,"low":0.06708265,"open":0.06711111,"close":0.06761028,"volume":154.42161693,"quoteVolume":2274.32121804,"weightedAverage":0.06789789},
{"date":1623513600,"high":0.06780281,"low":0.06726947,"open":0.06754057,"close":0.0674337,"volume":91.59523241,"quoteVolume":1357.82952572,"weightedAverage":0.06745709},
{"date":1623528000,"high":0.06750205,"low":0.0666125,"open":0.06744384,"close":0.06672966,"volume":67.80335981,"quoteVolume":1012.4453282,"weightedAverage":0.06696989},
{"date":1623542400,"high":0.0675751,"low":0.0665,"open":0.066749,"close":0.06695319,"volume":141.56780628,"quoteVolume":2109.77971741,"weightedAverage":0.06710075},
{"date":1623556800,"high":0.06698899,"low":0.06584509,"open":0.06685611,"close":0.06585844,"volume":15.59134405,"quoteVolume":233.91817861,"weightedAverage":0.06665298},
{"date":1623571200,"high":0.06630568,"low":0.06572449,"open":0.06582293,"close":0.06584084,"volume":9.31203953,"quoteVolume":141.14176333,"weightedAverage":0.06597649},
{"date":1623585600,"high":0.06608821,"low":0.0651492,"open":0.06584648,"close":0.06524958,"volume":130.93429314,"quoteVolume":2001.45813966,"weightedAverage":0.06541945},
{"date":1623600000,"high":0.06528874,"low":0.0644,"open":0.06517336,"close":0.06453459,"volume":105.23537129,"quoteVolume":1622.21097393,"weightedAverage":0.06487156},
{"date":1623614400,"high":0.0651,"low":0.06352759,"open":0.06453323,"close":0.06421791,"volume":71.22845226,"quoteVolume":1104.06555117,"weightedAverage":0.06451469},
{"date":1623628800,"high":0.06438708,"low":0.06325,"open":0.06430367,"close":0.0640978,"volume":81.34039153,"quoteVolume":1275.61037543,"weightedAverage":0.06376585}]

https://poloniex.com/public?command=returnChartData&currencyPair=BTC_ETH&start=1455699200&end=9999999999&period=14400

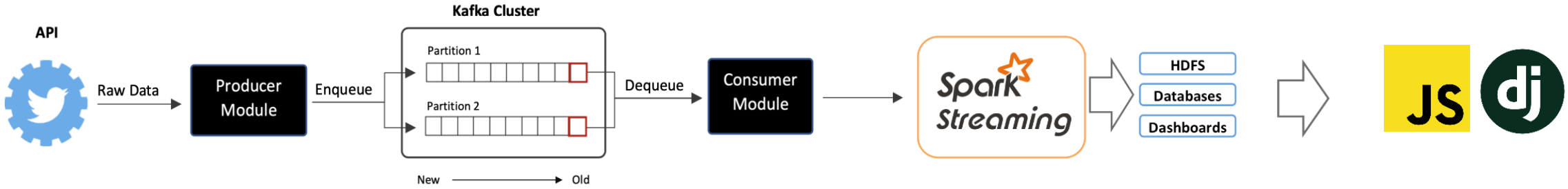
Type of Cryptocurrency

Start time

Real time update
Time period
300 -5min, 14400 – 4hr

2. Framework

Framework



Kafka
Data ingestion

Spark streaming
Data processing

MongoDB
Data storage

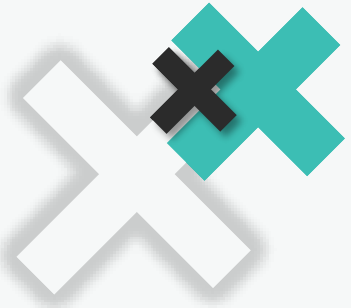
JavaScript
DJango

3. Detailed Description

Description *Twitter Streaming*

```
class twitterAuth():
        """SET UP TWITTER AUTHENTICATION"""
        def authenticateTwitterApp(self):
            auth = OAuthHandler(consumer_key, consumer_secret)
            auth.set_access_token(access_token, access_token_secret)
            return auth
```

Access to twitter streaming API with access_tokens



Description

Twitter Streaming

```
class TwitterStreamer():
        """SET UP STREAMER"""
        def __init__(self):
            self.twitterAuth = twitterAuth()

        def stream_tweets(self):
            while True:
                listener = ListenerTS()
                auth = self.twitterAuth.authenticateTwitterApp()
                stream = Stream(auth, listener)
                stream.filter(track=['bitcoin','BTC','bitcoin.com', W
                              'ETH', 'Ethereum', 'ethereum.org'])
```

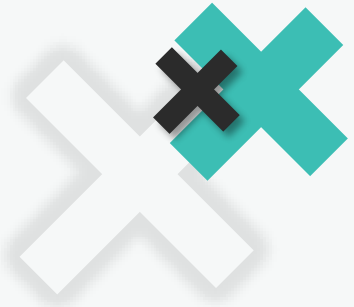
Listening the real time tweet stream and fillter it with predefined keywords

Description

Twitter Streaming

```
class ListenerTS(StreamListener):

    def on_data(self, raw_data):
        producer.send(topic_name, raw_data.encode('utf-8'))
        return True

if __name__ == "__main__":
    TS = TwitterStreamer()
    TS.stream_tweets()
```

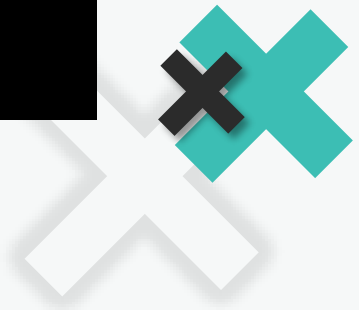
Create Twitter Streamer instance and stream tweets into kafka producer



Description

Apache Kafka

```
root@kafka1:~/kafka# bin/kafka-topics.sh --list --zookeeper kafka1:2181,kafka2:2181,kafka3:21
81/twitter
__consumer_offsets
test
test2
test3
root@kafka1:~/kafka# bin/kafka-topics.sh --describe --zookeeper kafka1:2181,kafka2:2181,kafka
3:2181/twitter --topic test2
Topic:test2     PartitionCount:10       ReplicationFactor:3     Configs:
        Topic: test2    Partition: 0    Leader: 3       Replicas: 3,2,1 Isr: 2,3,1
        Topic: test2    Partition: 1    Leader: 1       Replicas: 1,3,2 Isr: 2,3,1
        Topic: test2    Partition: 2    Leader: 2       Replicas: 2,1,3 Isr: 2,3,1
        Topic: test2    Partition: 3    Leader: 3       Replicas: 3,1,2 Isr: 2,3,1
        Topic: test2    Partition: 4    Leader: 1       Replicas: 1,2,3 Isr: 2,3,1
        Topic: test2    Partition: 5    Leader: 2       Replicas: 2,3,1 Isr: 2,3,1
        Topic: test2    Partition: 6    Leader: 3       Replicas: 3,2,1 Isr: 2,3,1
        Topic: test2    Partition: 7    Leader: 1       Replicas: 1,3,2 Isr: 2,3,1
        Topic: test2    Partition: 8    Leader: 2       Replicas: 2,1,3 Isr: 2,3,1
        Topic: test2    Partition: 9    Leader: 3       Replicas: 3,1,2 Isr: 2,3,1
```



Create kafka topic named ‘test2’ with 10 partitions

Description

Sentimental Analysis

PySpark

NLTK

```
I am happy --> 1.0  Sad day.. --> -1
```


```
Christine Lagarde Reaffirms ECB's Crypto Policy
 as #Bitcoin Becomes Legal Tender  in El Salvador
 #Btc #Cryptocurrency #Finance #Forex --> 1.0
```

Positive


```
I had some bitcoin but due to a horrible COVID
 accident I lost the keys to my wallet --> -1
```

Negative

Description

Sentimental Analysis

```
def regex_(text):
        pattern = '(http|ftp|https)://(?:[-₩w.]|(?:%[₩da-fA-F]{2}))+/(?:[-₩w.]|(?:%[₩da-fA-F]
{2}))+'
        text = re.sub(pattern, '', text)
        pattern = '(http|ftp|https)://(?:[-₩w.]|(?:%[₩da-fA-F]{2}))+'
        text = re.sub(pattern, '', text)
        pattern = '(http|ftp|https):// (?:[-₩w.]|(?:%[₩da-fA-F]{2}))+'
        text = re.sub(pattern, '', text)
        only_english = re.sub('[^ a-zA-Z]', '', text)
        only_english = only_english.lower()

        if bool(only_english and only_english.strip()) and len(only_english) >= 10:
            return only_english

        return text
```

Pre-processing the tweet text for sentiment analysis (URL, Non-English elements, lower character)

Description

Sentimental Analysis

```
def nlkt_analysis(text):

    if(type(text) != type('str')): return

    try:
        sia = SentimentIntensityAnalyzer()
        text = regex_(text)
        temp = sia.polarity_scores(text)["compound"]

        if temp == 0.0:
            return temp
        elif temp > 0.0:
            return 1.0
        else:
            return -1.0

    except Exception as e:
        print(e)
        return e
```

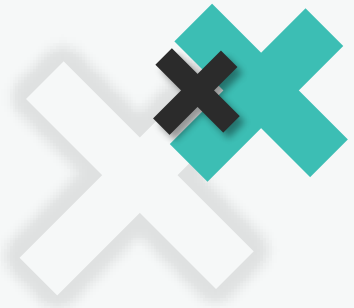
Using the NLTK, analyze the emotional polarity of tweet text and Divide into 3 categories (POS, NEG, NEU)

Description

Spark Structured Streaming

```
if __name__ == "__main__":
    spark = SparkSession ₩
            .builder ₩
            .appName("TwitterStreamingAPI") ₩
            .config("spark.mongodb.input.uri","mongodb://117.17.189.6:27017/tweet.kafka_tweet")₩
            .config("spark.mongodb.output.uri","mongodb://117.17.189.6:27017/tweet.kafka_tweet")₩
            .getOrCreate()

    spark.sparkContext.setLogLevel("ERROR")

    kafka_df = spark.readStream.format("kafka").option("partition.assignment.strategy","range").option("kafka.bootstrap.server
s", "localhost:9092").option("subscribe", "test").option("startingOffsets", "latest").load()
    kafka_df.printSchema()
    kafka_df1 = kafka_df.selectExpr("CAST(value AS STRING)", "timestamp")
```

Make a sparkSession to corresponding address of database and build the readStream for kafka topic

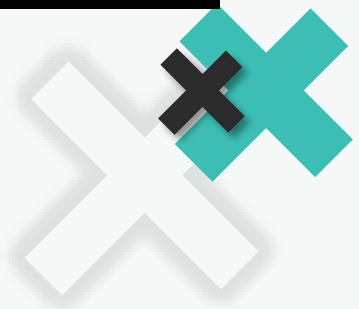


Description

Spark Structured Streaming

```
transaction_detail_schema = StructType() ₩
        .add("id", StringType()) ₩
        .add("text", StringType()) ₩
        .add("created_at", StringType())₩

kafka_df2 = kafka_df1 ₩
        .select(from_json(col("value"), transaction_detail_schema).alias("transaction_detail"), "timestamp")

kafka_df3 = kafka_df2.select("transaction_detail.*", "timestamp")

udf_blob = udf(apply_blob, StringType())

udf_nlkt = udf(nlkt_analysis, StringType())

new_df = kafka_df3.withColumn("status", udf_nlkt("text"))
new_df = new_df.drop("text")
```

Make a RDD StructType with 3 columns including “id”, “created_at”, “timestamp”, “status”

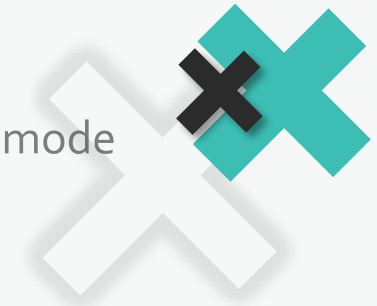


Description

Spark Structured Streaming

```
def write_mongo_row(df, epoch_id):
    mongoURL = "mongodb://117.17.189.6:27017/tweet.kafka_tweet"
    df.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").option("uri", mongoURL).save()
    df.show()
    pass
query = new_df.writeStream.trigger(processingTime='1 seconds').foreachBatch(write_mongo_row).start()
query.awaitTermination()

spark.stop()
```

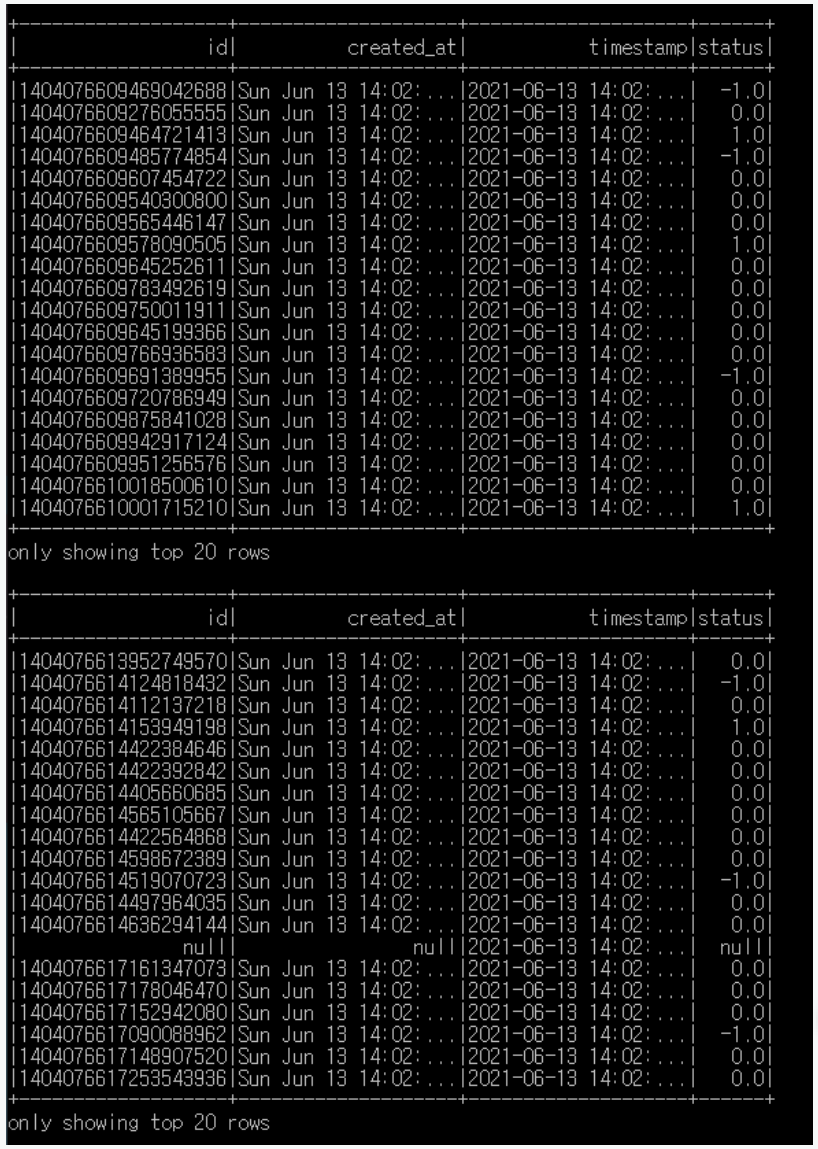
ForeachBatch with predefined processing Time, we store it into mongodb with append mode



Description

Final Console

The final Dataframe in Spark streaming is stored in mongoDB with each Batch



```
+-------------------+-------------------+-------------------+------+
|                 id|         created_at|          timestamp|status|
+-------------------+-------------------+-------------------+------+
|1404076609469042688|Sun Jun 13 14:02:...|2021-06-13 14:02:...|  -1.0|
|1404076609276055555|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076609464721413|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   1.0|
|1404076609485774854|Sun Jun 13 14:02:...|2021-06-13 14:02:...|  -1.0|
|1404076609607454722|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076609540300800|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076609565446147|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076609578090505|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   1.0|
|1404076609645252611|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076609783492619|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076609750011911|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076609645199366|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076609766936583|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076609691389955|Sun Jun 13 14:02:...|2021-06-13 14:02:...|  -1.0|
|1404076609720786949|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076609875841028|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076609942917124|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076609951256576|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076610018500610|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076610001715210|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   1.0|
+-------------------+-------------------+-------------------+------+
only showing top 20 rows

+-------------------+-------------------+-------------------+------+
|                 id|         created_at|          timestamp|status|
+-------------------+-------------------+-------------------+------+
|1404076613952749570|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076614124818432|Sun Jun 13 14:02:...|2021-06-13 14:02:...|  -1.0|
|1404076614112137218|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076614153949198|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   1.0|
|1404076614422384646|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076614422392842|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076614405660685|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076614565105667|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076614422564868|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076614598672389|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076614519070723|Sun Jun 13 14:02:...|2021-06-13 14:02:...|  -1.0|
|1404076614497964035|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076614636294144|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|               null|               null|2021-06-13 14:02:...|  null|
|1404076617161347073|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076617178046470|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076617152942080|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076617090088962|Sun Jun 13 14:02:...|2021-06-13 14:02:...|  -1.0|
|1404076617148907520|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
|1404076617253543936|Sun Jun 13 14:02:...|2021-06-13 14:02:...|   0.0|
+-------------------+-------------------+-------------------+------+
only showing top 20 rows
```

Description

Mongo DB

The sentiment analysis data is stored in mongoDB

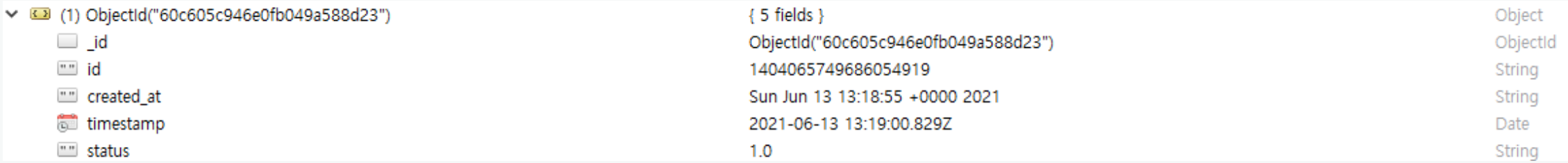
kafka_tweet 0,012 sec.

Key	Value	Type
(1) ObjectId("60c605c946e0fb049a588d23")	{ 5 fields }	Object
_id	ObjectId("60c605c946e0fb049a588d23")	ObjectId
id	1404065749686054919	String
created_at	Sun Jun 13 13:18:55 +0000 2021	String
timestamp	2021-06-13 13:19:00.829Z	Date
status	1.0	String
(2) ObjectId("60c605c946e0fb049a588d24")	{ 5 fields }	Object
_id	ObjectId("60c605c946e0fb049a588d24")	ObjectId
id	1404065749627506691	String
created_at	Sun Jun 13 13:18:55 +0000 2021	String
timestamp	2021-06-13 13:19:00.832Z	Date
status	0.0	String
(3) ObjectId("60c605c946e0fb049a588d25")	{ 5 fields }	Object
_id	ObjectId("60c605c946e0fb049a588d25")	ObjectId
id	1404065749837119489	String
created_at	Sun Jun 13 13:18:55 +0000 2021	String
timestamp	2021-06-13 13:19:00.856Z	Date
status	0.0	String
(4) ObjectId("60c605c946e0fb049a588d26")	{ 5 fields }	Object
_id	ObjectId("60c605c946e0fb049a588d26")	ObjectId
id	1404065749870600194	String
created_at	Sun Jun 13 13:18:55 +0000 2021	String
timestamp	2021-06-13 13:19:00.883Z	Date
status	1.0	String
(5) ObjectId("60c605c946e0fb049a588d27")	{ 5 fields }	Object
_id	ObjectId("60c605c946e0fb049a588d27")	ObjectId
id	1404065750013206537	String
created_at	Sun Jun 13 13:18:55 +0000 2021	String
timestamp	2021-06-13 13:19:00.953Z	Date
status	1.0	String
(6) ObjectId("60c605c946e0fb049a588d28")	{ 5 fields }	Object
_id	ObjectId("60c605c946e0fb049a588d28")	ObjectId
id	1404065750042611716	String
created_at	Sun Jun 13 13:18:55 +0000 2021	String
timestamp	2021-06-13 13:19:00.980Z	Date
status	0.0	String

Description

Mongo DB

Specific columns explanation



- **Id:** Twitter user ID
- **Created_at:** Timestamp when user uploaded tweet
- **Timestamp:** Timestamp when tweet data saved at mongoDB
- **Status:** Sentimental analysis result for tweet text data

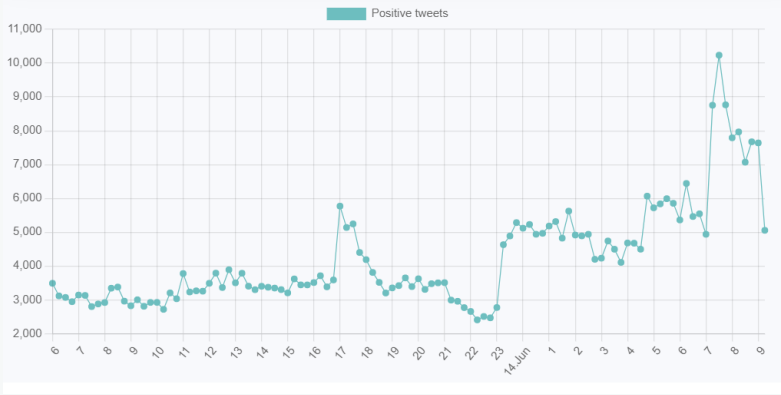
Description

Web programming

```
$.getJSON('https://poloniex.com/public?command=returnChartData&currencyPair=USDT_BTC&start=1623533200&end=9999999999&period=900', function (data)
    $.each(data, function(i, item){
        chartdata.push([item.date*1000, item.open, item.high, item.low, item.close]);
    });
}).done(function(){
    Highcharts.stockChart('bitcoin',{
        title: {
            text: 'USDT_BTC'
        },
```

<get json data and make graph>


```
var myChart = new Chart(ctx, {
    type: 'line',
    data: {
        labels: temp2,

        datasets: [{
            borderColor: 'rgb(75, 192, 192)',
            backgroundColor: 'rgb(75, 192, 192)',
            label: 'Positive tweets',
            data: temp1,
            borderWidth: 1
        }]
    },
```



<get sentimental data and make graph>

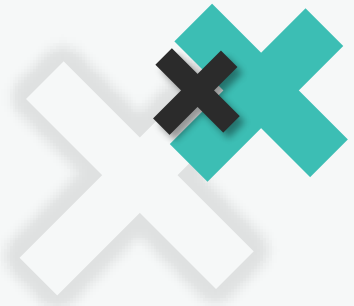
Description

Web programming

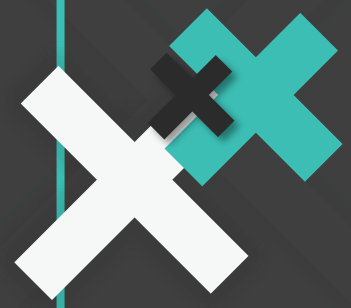
```
class setimentalResult(object):
    def __init__(self):
        client = MongoClient("117.17.189.6", 27017)
        self.db = client['tweet']
        self.collection = self.db.kafka_tweet_2

    def get_users_from_collection(self):
        doc=self.collection.find({})
        return doc
```

<get data from mongoDB>

```
for tweet in results:
    str_datetime = "2021-06-" + tweet['created_at'][8:10] + " " + tweet['created_at'][11:19]
    convert = datetime.datetime.strptime(str_datetime, dateformat)
    if tweet['sent_result'] == 1.0:
        count += 1
    if convert >= datetime.datetime.fromtimestamp(startTimestamp + 900):
        date.append(startTimestamp)
        result.append(count)
        startTimestamp += 900
        count = 0
return render(request, 'index.html', {"date":date, "result": result})
```



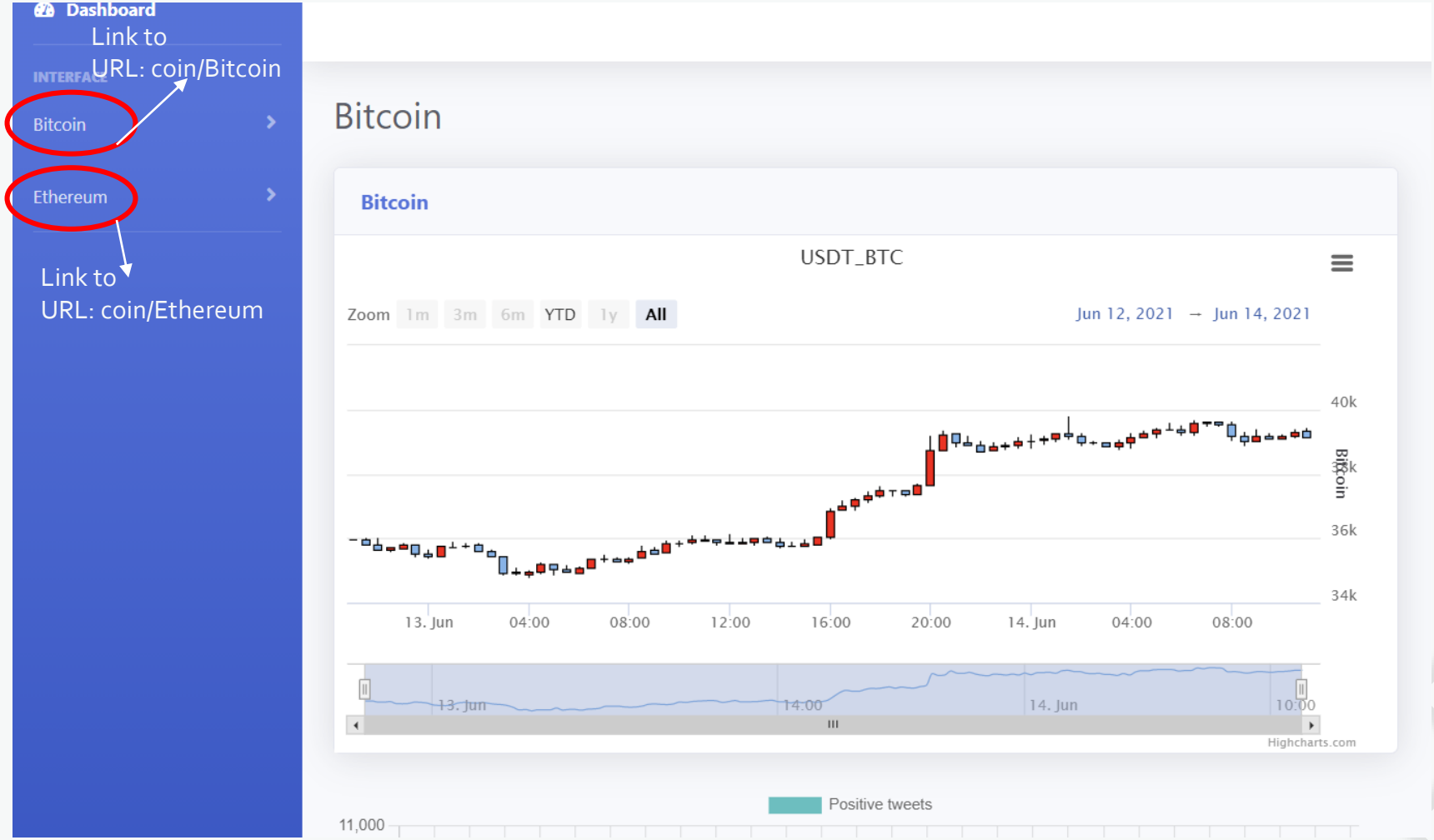
<Count positive tweets for every 15min.>



4. Final Results

Results

Web page (main)

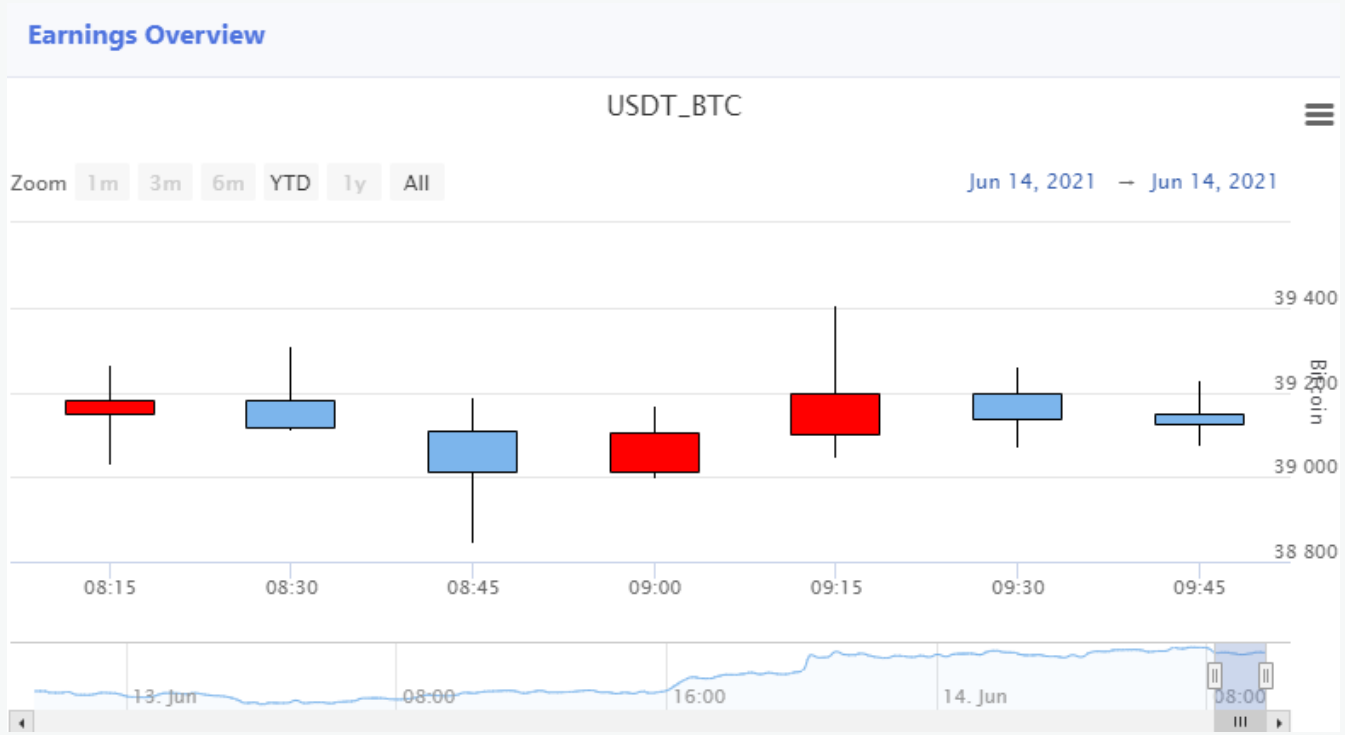


Results

Price chart

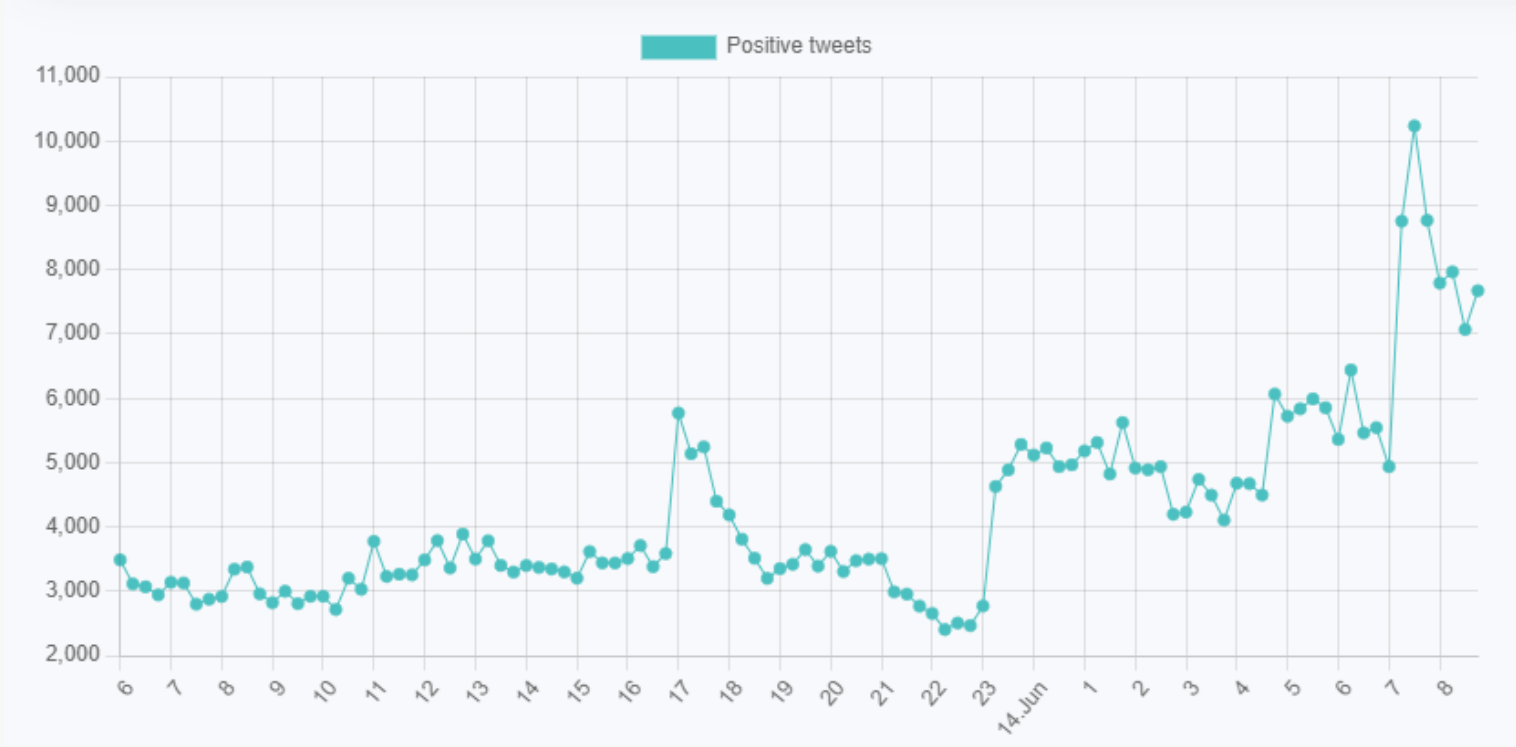
Results

Price chart with different time interval



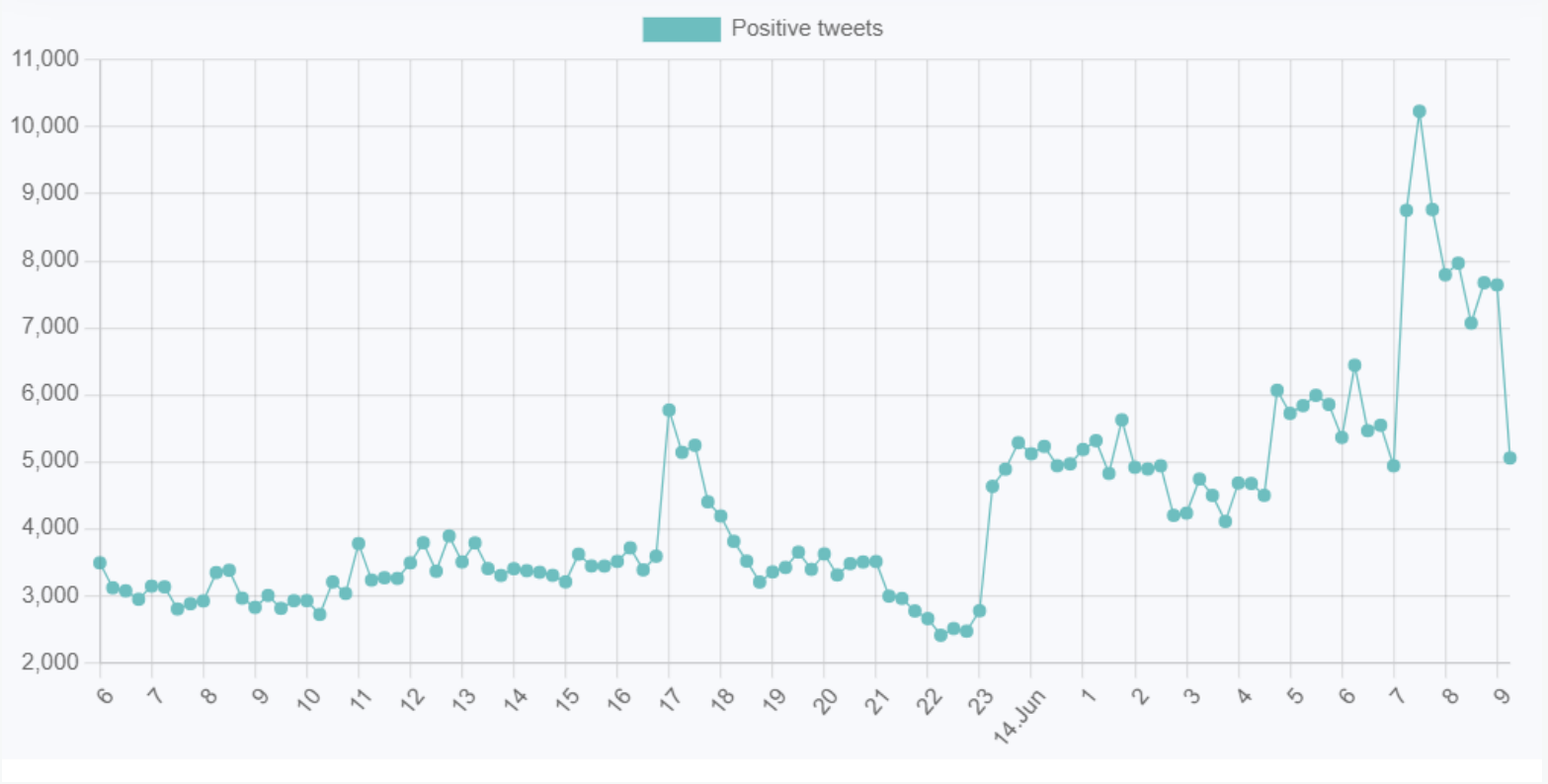
Results

Chart result of sentimental analysis



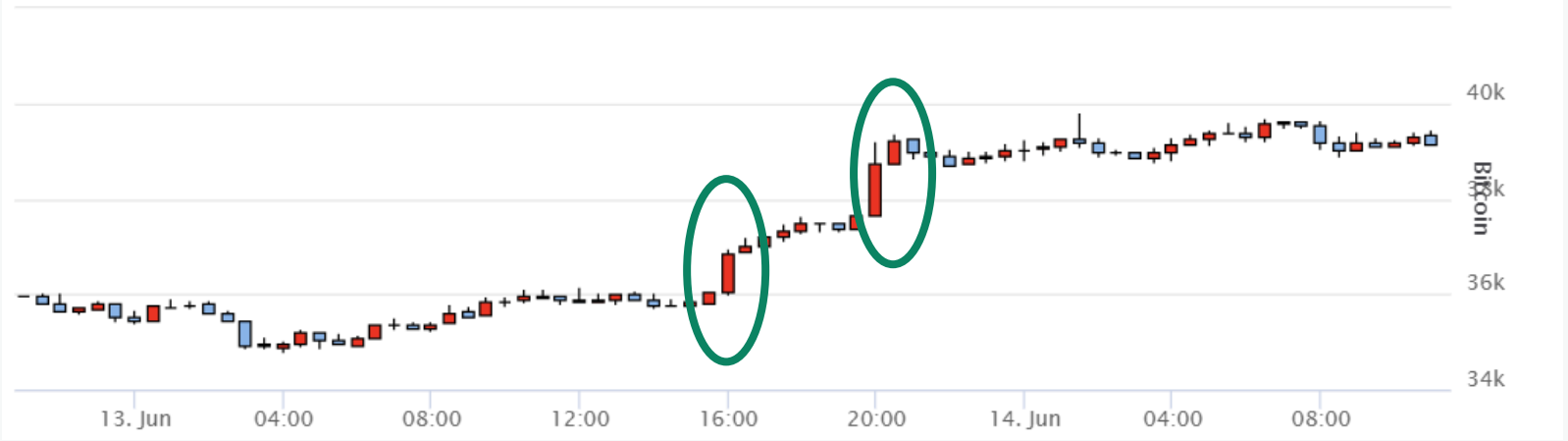
Results

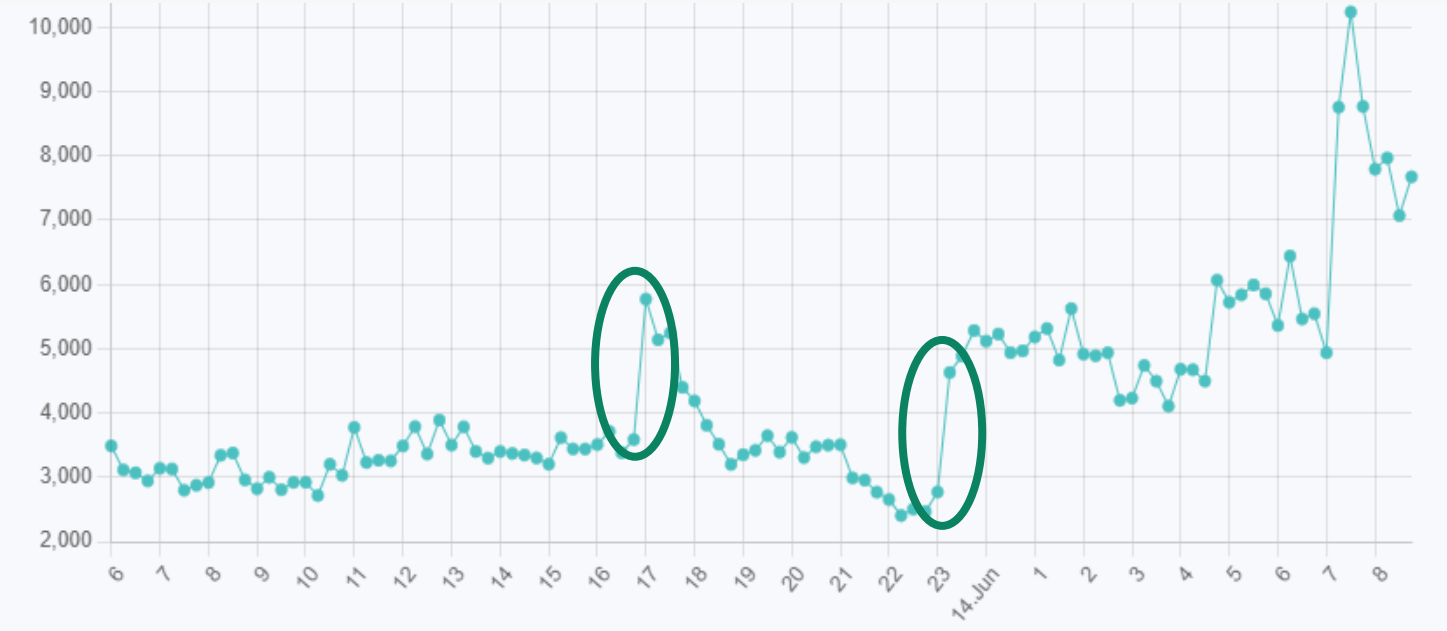
Updates dynamically for every 15 mins.



Results

Correlation between price and sentimental

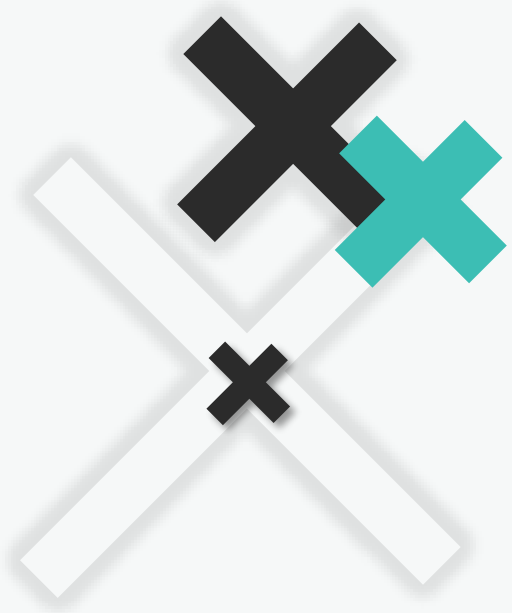




5. Problems

Problems

Project implementation



- **Port Forwarding**
- **Dependency problems**
  due to version mismatch
- **MongoDB external access**
- **Input database for each RDD**
  foreachbatch method

Thank you!