1902224

Set the size of shuffles to 1, in order to prevent Spark from over partitioning the data:

```
from pyspark.sql.functions import *

spark.conf.set("spark.sql.shuffle.partitions", "1")
```

Set up raw streaming DataFrame by connecting this DataFrame to my Kinesis stream:

- set credentials
- create data frame

```
awsAccessKey = "[redacted]"
awsSecretKey = "[redacted]"
kinesisStreamName = "KS-1902224"
kinesisRegion = "eu-west-1" # Dublin
```

```
rawKinesisDF = (spark.readStream
  .format("kinesis")
  .option("streamName", kinesisStreamName)
  .option("region", kinesisRegion)
  .option("initialPosition", "latest") # <---- SETTING THE "offset".
  .option("awsAccessKey", awsAccessKey)
  .option("awsSecretKey", awsSecretKey)
  .load())
```

Print the schema of the resulting DataFrame:

```
rawKinesisDF.printSchema()
```

```
root
 |-- partitionKey: string (nullable = true)
 |-- data: binary (nullable = true)
 |-- stream: string (nullable = true)
```

```
 |-- shardId: string (nullable = true)
 |-- sequenceNumber: string (nullable = true)
 |-- approximateArrivalTimestamp: timestamp (nullable = true)
```

Decode the data column to its original value and show it in column decoded_data:

```
KinesisDF = rawKinesisDF.selectExpr("CAST(data as STRING) as decoded_data",
"approximateArrivalTimestamp as receipt_time")
display(KinesisDF.selectExpr("decoded_data"))
```

▸ display_query_21 (id: 93c2cd5d-b7b2-4e4c-951f-d6ba535c4646) *Last updated: 240 days ago*

decoded_data
{"e":"trade","event_time":1.583166738308E9,"s":"BTCUSDT","p":8873.800000000001,"q":0.001912
{"e":"trade","event_time":1.583166738629E9,"s":"BTCUSDT","p":8873.6,"q":0.053786,"m":false}
{"e":"trade","event_time":1.5831667386330001E9,"s":"BNBBTC","p":0.0022379,"q":0.2,"m":false}
{"e":"trade","event_time":1.583166738709E9,"s":"NEOUSDT","p":11.975,"q":20.25,"m":false}
{"e":"trade","event_time":1.5831667388270001E9,"s":"BTCUSDT","p":8873.42,"q":0.002252,"m":fal
{"e":"trade","event_time":1.583166739543E9,"s":"BTCUSDT","p":8871.97,"q":0.00825,"m":false}
{"e":"trade","event_time":1.583166739652E9,"s":"BTCUSDT","p":8871.800000000001,"q":0.003016
{"e":"trade","event_time":1.583166740414E9,"s":"LTCUSDT","p":61.25,"q":2.12691,"m":false}

Showing the first 1000 rows.

Create a schema object to this DataFrame: Keep EVENT_TIME, S, P and Q, but don't keep m, E and T.

```
from pyspark.sql.types import *

schema = StructType([
  StructField("event_time",DoubleType(), True),
  StructField("s", StringType(), True),
  StructField("p", DoubleType(), True),
  StructField("q", DoubleType(), True)
])
```

Apply the schema to your DataFrame. This is achieved by parsing the JSON values into a structured format and converted to top-level column (EVENT_TIME, S, P and Q).

```
from pyspark.sql.functions import from_json, col

parsedDF = KinesisDF.select(from_json(col("decoded_data"),
schema).alias("j"), col("receipt_time"))

trades = parsedDF.selectExpr("j.*")
display(trades)
```

display_query_22 (id: 2190df7c-78e2-49c8-9d81-55099cd03e9e) *Last updated: 240 days ago*

event_time	s	p
1583166742.8600001	BNBBTC	0.0
1583166743.315	BTCUSDT	88
1583166743.425	ETHUSDT	23
1583166743.491	ETHUSDT	23
1583166743.491	ETHUSDT	23
1583166743.491	ETHUSDT	23
1583166743.491	ETHUSDT	23
1583166743.491	ETHUSDT	23
1583166743.491	ETHUSDT	23

Showing the first 1000 rows.

Change dataframe format:

- Round event_time to the closest lowest integer (it’s called flooring) and convert it from unixtime to datetime format.
- Rename S to currency
- Rename P to price
- Rename Q to quantity
- Create a new column called trade_amount, which takes P*Q as its value
- Display the resulting DataFrame

```
trades2 = trades.select(
  to_timestamp(floor("event_time")).alias("event_time"),
  col("s").alias("currency"),
  col("p").alias("price"),
  col("q").alias("quantity")
              )
trades2 = trades2.withColumn("trade_amount",trades2.price*trades2.quantity)

display(trades2)
```

▸ display_query_23 (id: 81ad11ca-9057-4fe3-98b8-9909cb75457d) *Last updated: 240 days ago*

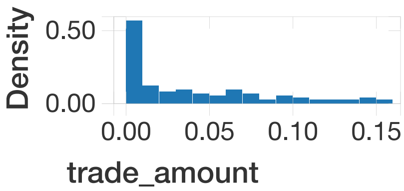
event_time	currency	price
2020-03-02T16:32:22.000+0000	BNBBTC	0.002236
2020-03-02T16:32:23.000+0000	BTCUSDT	8867.24
2020-03-02T16:32:23.000+0000	ETHUSDT	231.77
2020-03-02T16:32:23.000+0000	ETHUSDT	231.78
2020-03-02T16:32:23.000+0000	ETHUSDT	231.78
2020-03-02T16:32:23.000+0000	ETHUSDT	231.79
2020-03-02T16:32:23.000+0000	ETHUSDT	231.79
2020-03-02T16:32:23.000+0000	ETHUSDT	231.8
2020-03-02T16:32:23.000+0000	ETHUSDT	231.81

Showing the first 1000 rows.

Display a histogram plot of the trade_amounts of transactions where Ethereum (ETH) is sold to buy Bitcoin (BTC). Keep it running for at least a minute so see some patterns emerge.

```
display(trades2.filter("currency = 'ETHBTC'").select("trade_amount"))
```

▸ display_query_28 (id: 15fe7ec2-2305-4295-b84d-61088f8e7dcc) *Last updated: 240 days ago*



Top 5 currency pairs, ordered by sum(trade_amount).

```
top5curr =
trades2.groupBy("currency").sum("trade_amount").orderBy(col("sum(trade_amoun
t)").desc())
display(top5curr.limit(5))
```

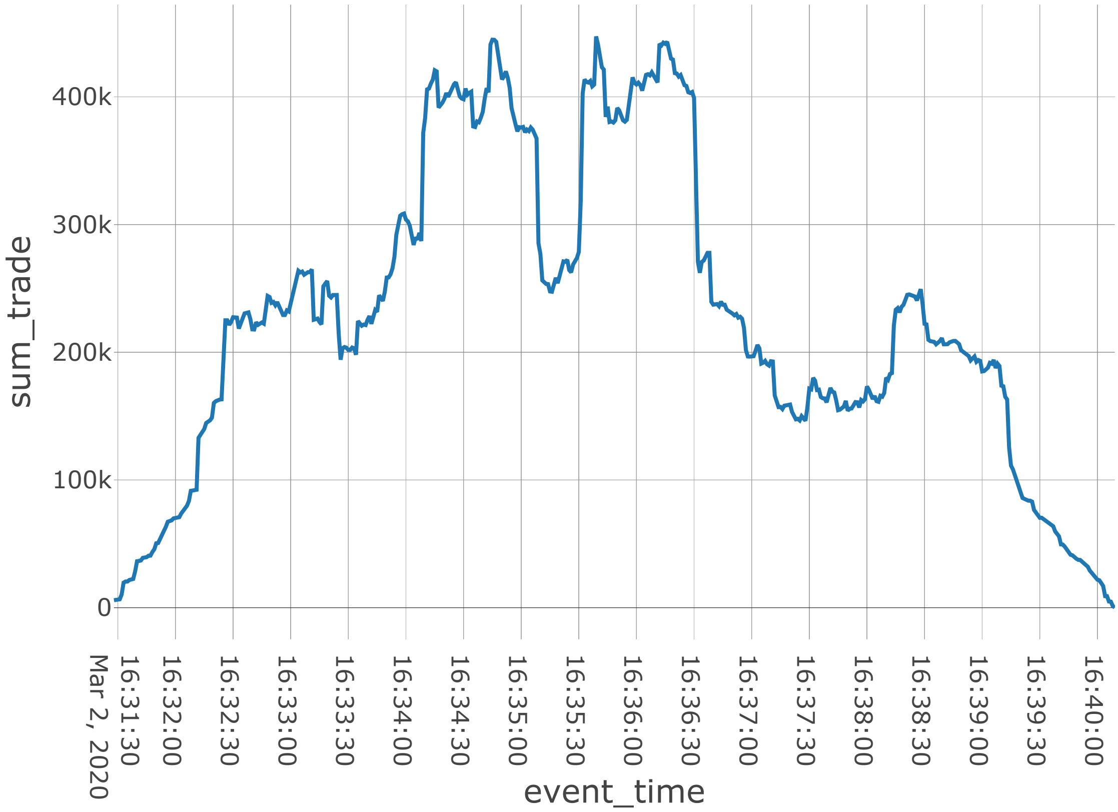
display_query_25 (id: 8a14daa0-28a7-46ed-8fdc-8c61a795b72e) *Last updated: 240 days ago*

currency	sum(trade_amou
BTCUSDT	1061033.3048365
ETHUSDT	570596.5573053
BNBUSDT	185405.5940439
LTCUSDT	127387.4574929
NEOUSDT	26585.84849700

Line chart that displays the rolling sum of trade_amounts (1-minute window interval, 1-second slide interval)

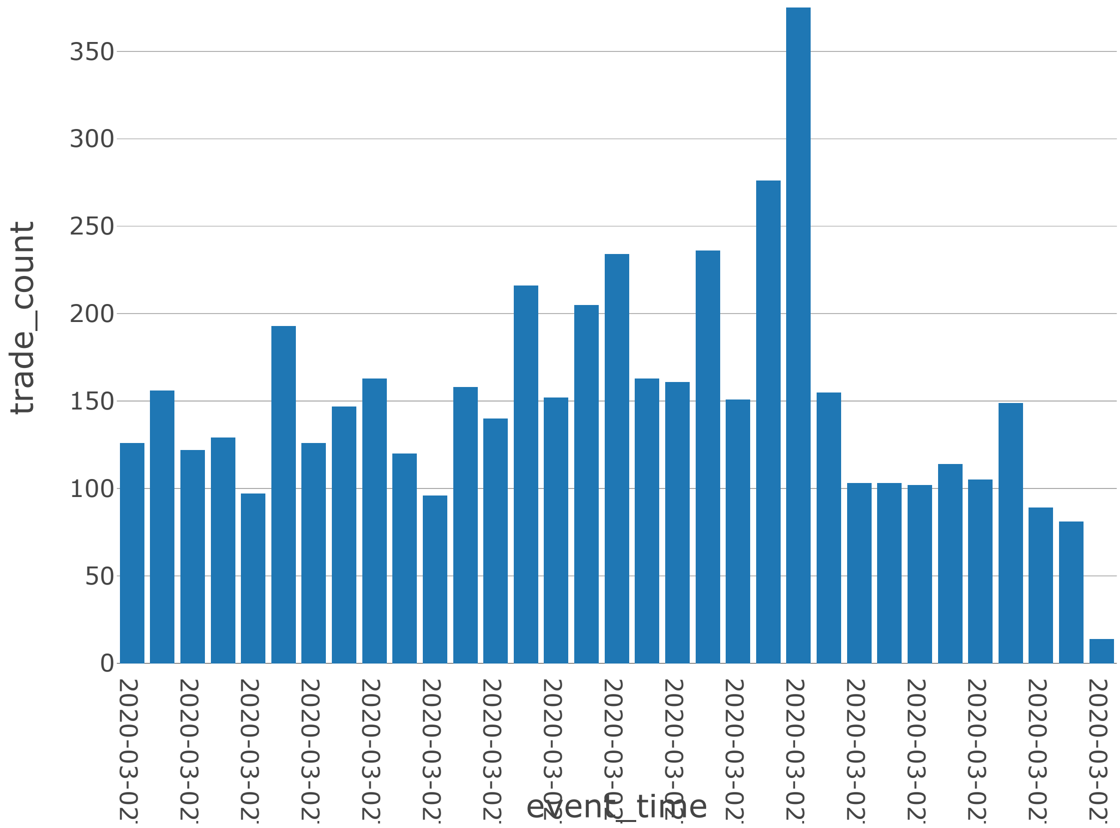
```
display(trades2
 .groupBy(window(col("event_time"), '1 minute', '1
second')).sum("trade_amount").withColumnRenamed("sum(trade_amount)","sum_tra
de")
 .orderBy(col("window.start").desc())

.selectExpr("window.start","sum_trade").withColumnRenamed("start","event_tim
e")
)
```

display_query_26 (id: 4db294da-5d5f-4872-b4c0-82112dff880b) *Last updated: 240 days ago*



Bar chart that displays the number of trade records for every 15 seconds, juxtaposed (no overlapping windows).

display_query_27 (id: af049275-3c28-410c-a052-50a3ee8c949e) *Last updated: 240 days ago*



trade_count
0
50
100
150
200
250
300
350
event_time
2020-03-02