Big Data Assignment

PART 1: Kafka

There are 2 scripts used to make a connection - producer and consumer.

The **producer script** connects to Bianance's WebSocket API and receives the trade events and writes them to a kafka topic (btcusdt_trades).

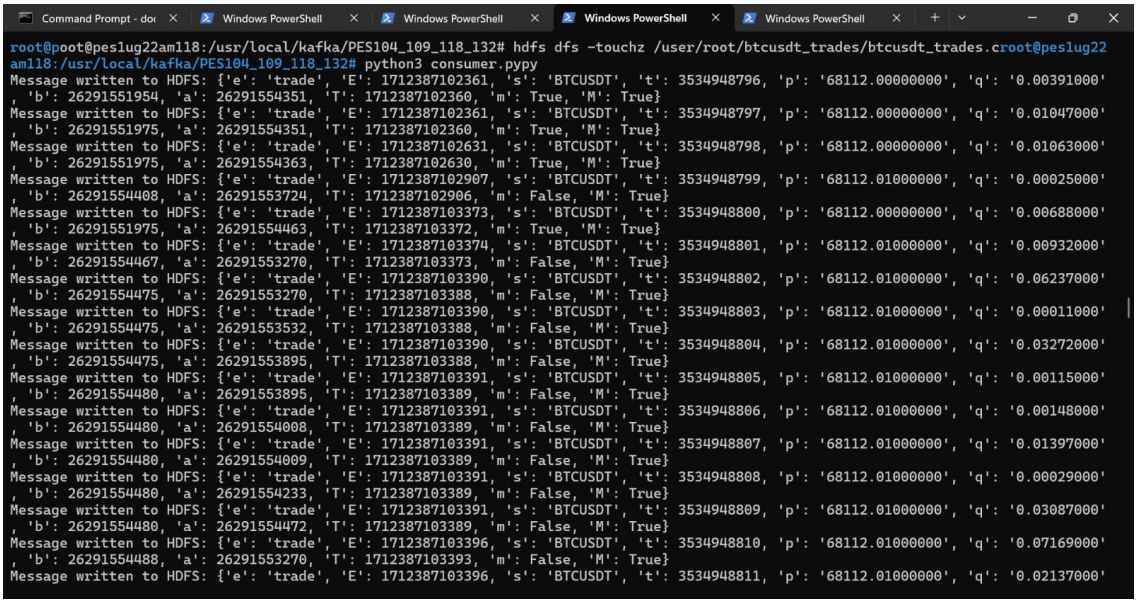
```
# Wingardium Leviosa! Importing some magical libraries
from kafka import KafkaProducer
from websocket import create_connection  # Connecting to the w
import json


async def main():
    # Summon our trusty Kafka producer
    producer = KafkaProducer(bootstrap_servers='localhost:909
    # Let's name our topic 'btcusdt_trades' for safekeeping
    topic = 'btcusdt_trades'

    # Revelio! Uncover the hidden treasures of the WebSocket w
    ws_url = "wss://stream.binance.com:9443/ws/btcusdt@trade"
    # Open the portal to the WebSocket realm
    ws = create_connection(ws_url)

    # Let's disarm incoming messages and see what they hold!
    while True:
        # Incoming message alert! Received a message from the
        message = ws.recv()
        # Sending that message off to our secure Kafka topic
        producer.send(topic, value=message.encode())
        # Message sent! Let's keep an eye on the console for
        print("message sent to kafka", message)
        # Clean up the buffer, gotta keep things efficient.
        producer.flush()

# Mischief managed! Let's make sure we run this script correc
```

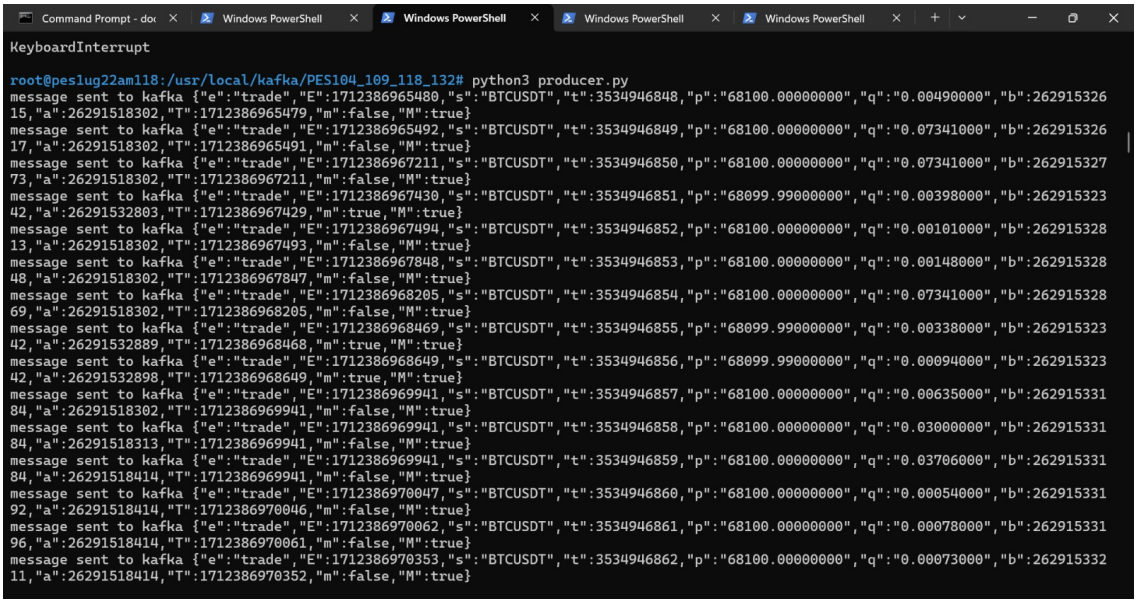
```
if _ _name_ _== " _ _main_ _":
    # Accio asyncio! Import it to run the magic asynchronousl
    import asyncio
    asyncio.run(main())
```

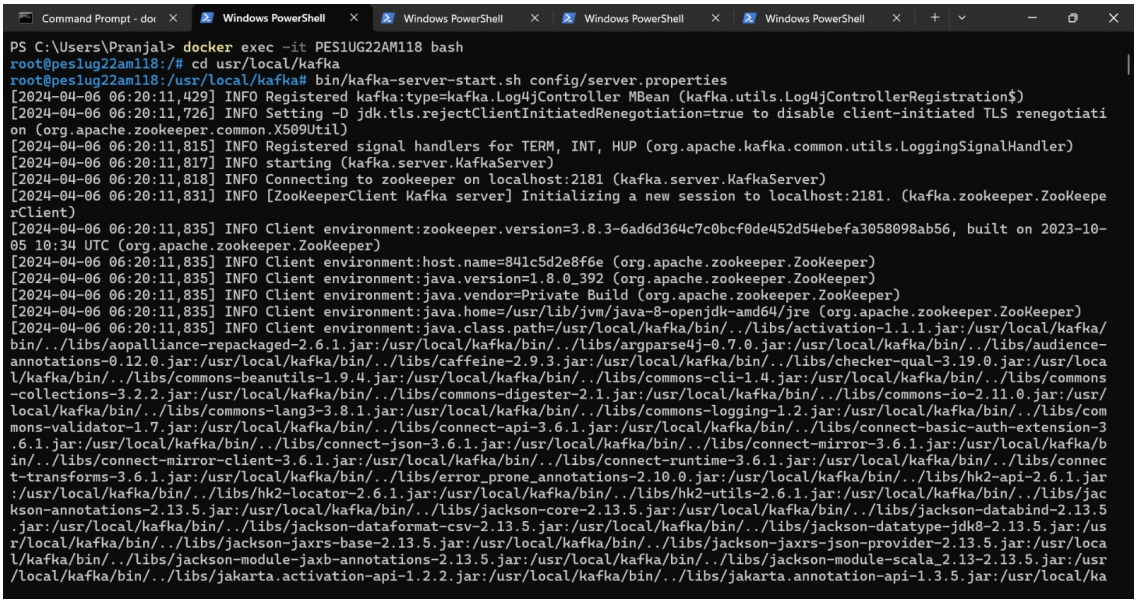
The **consumer script** connects to the topic (btcusdt_trades), takes in the messages, processes then writes it to HDFS in csv format.

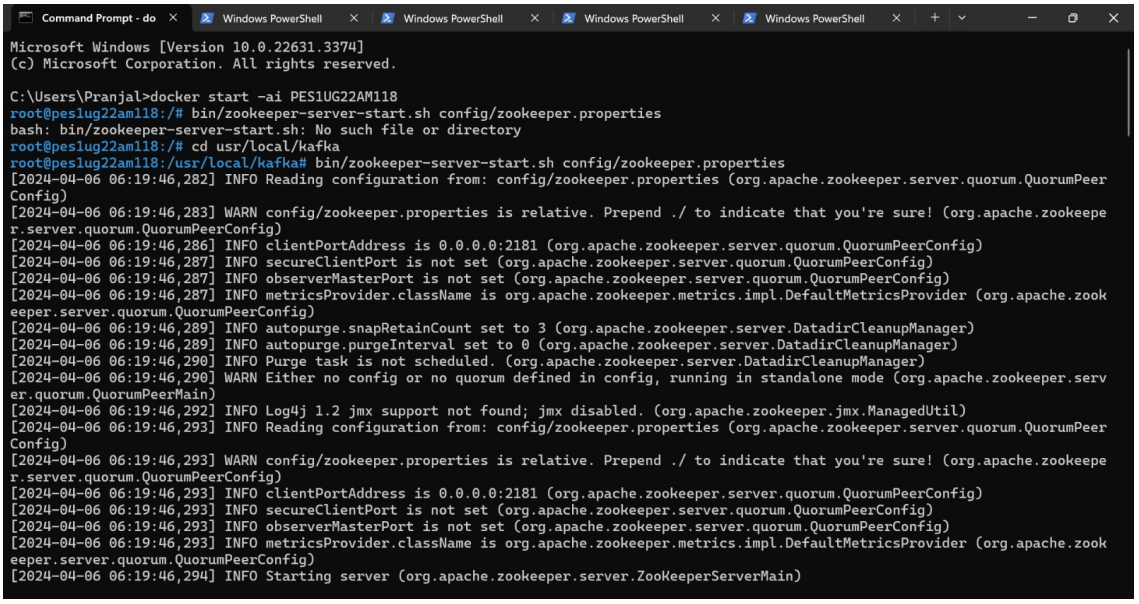
```
#!/bin/bash/env python3
from kafka import KafkaConsumer  # Efficient message capture,
from hdfs import InsecureClient  # Secure HDFS access, gotta
import json  # JSON parsing, elegance in data representation

def main():

    #The heart of the operation - ingests trade data and orch
    consumer = KafkaConsumer('btcusdt_trades', bootstrap_serv
    hdfs_client = InsecureClient('http://localhost:9870', use

    for message in consumer:
        #Iterate through incoming messages, a continuous stre
        trade_event = json.loads(message.value.decode('utf-8'
        write_to_hdfs(hdfs_client, trade_event)
        print("Message written to HDFS:", trade_event)

def write_to_hdfs(client, trade_event):
    #Writes trade data to HDFS, a robust and scalable storage
    #Think of it as a meticulously organized digital vault.
    with client.write('/user/root/btcusdt_trades/btcusdt_trad
        writer.write(json.dumps(trade_event) + '\n')

if _ _name_ _ == "_ _main_ _":
    main()
```

OUTPUT


Command Prompt - doc
Windows PowerShell
Windows PowerShell
Windows PowerShell
Windows PowerShell
root@poot@pes1ug22am118:/usr/local/kafka/PES104_109_118_132# hdfs dfs -touchz /user/root/btcusdt_trades/btcusdt_trades.croot@pes1ug22am118:/usr/local/kafka/PES104_109_118_132# python3 consumer.pypy
Message written to HDFS: {'e': 'trade', 'E': 1712387102361, 's': 'BTCUSDT', 't': 3534948796, 'p': '68112.00000000', 'q': '0.00391000', 'b': 26291551954, 'a': 26291554351, 'T': 1712387102360, 'm': True, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387102361, 's': 'BTCUSDT', 't': 3534948797, 'p': '68112.00000000', 'q': '0.01047000', 'b': 26291551975, 'a': 26291554351, 'T': 1712387102360, 'm': True, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387102631, 's': 'BTCUSDT', 't': 3534948798, 'p': '68112.00000000', 'q': '0.01063000', 'b': 26291551975, 'a': 26291554363, 'T': 1712387102630, 'm': True, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387102907, 's': 'BTCUSDT', 't': 3534948799, 'p': '68112.01000000', 'q': '0.00025000', 'b': 26291554408, 'a': 26291553724, 'T': 1712387102906, 'm': False, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387103373, 's': 'BTCUSDT', 't': 3534948800, 'p': '68112.00000000', 'q': '0.00688000', 'b': 26291551975, 'a': 26291554463, 'T': 1712387103372, 'm': True, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387103374, 's': 'BTCUSDT', 't': 3534948801, 'p': '68112.01000000', 'q': '0.00932000', 'b': 26291554467, 'a': 26291553270, 'T': 1712387103373, 'm': False, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387103390, 's': 'BTCUSDT', 't': 3534948802, 'p': '68112.01000000', 'q': '0.06237000', 'b': 26291554475, 'a': 26291553270, 'T': 1712387103388, 'm': False, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387103390, 's': 'BTCUSDT', 't': 3534948803, 'p': '68112.01000000', 'q': '0.00011000', 'b': 26291554475, 'a': 26291553532, 'T': 1712387103388, 'm': False, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387103390, 's': 'BTCUSDT', 't': 3534948804, 'p': '68112.01000000', 'q': '0.03272000', 'b': 26291554475, 'a': 26291553895, 'T': 1712387103388, 'm': False, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387103391, 's': 'BTCUSDT', 't': 3534948805, 'p': '68112.01000000', 'q': '0.00115000', 'b': 26291554480, 'a': 26291553895, 'T': 1712387103389, 'm': False, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387103391, 's': 'BTCUSDT', 't': 3534948806, 'p': '68112.01000000', 'q': '0.00148000', 'b': 26291554480, 'a': 26291554008, 'T': 1712387103389, 'm': False, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387103391, 's': 'BTCUSDT', 't': 3534948807, 'p': '68112.01000000', 'q': '0.01397000', 'b': 26291554480, 'a': 26291554009, 'T': 1712387103389, 'm': False, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387103391, 's': 'BTCUSDT', 't': 3534948808, 'p': '68112.01000000', 'q': '0.00029000', 'b': 26291554480, 'a': 26291554233, 'T': 1712387103389, 'm': False, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387103391, 's': 'BTCUSDT', 't': 3534948809, 'p': '68112.01000000', 'q': '0.03087000', 'b': 26291554480, 'a': 26291554472, 'T': 1712387103389, 'm': False, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387103396, 's': 'BTCUSDT', 't': 3534948810, 'p': '68112.01000000', 'q': '0.07169000', 'b': 26291554488, 'a': 26291553270, 'T': 1712387103393, 'm': False, 'M': True}
Message written to HDFS: {'e': 'trade', 'E': 1712387103396, 's': 'BTCUSDT', 't': 3534948811, 'p': '68112.01000000', 'q': '0.02137000'


Command Prompt - doc
Windows PowerShell
Windows PowerShell
Windows PowerShell
Windows PowerShell
KeyboardInterrupt
root@pes1ug22am118:/usr/local/kafka/PES104_109_118_132# python3 producer.py
message sent to kafka {"e":"trade","E":1712386965480,"s":"BTCUSDT","t":3534946848,"p":"68100.00000000","q":"0.00490000","b":26291532615,"a":26291518302,"T":1712386965479,"m":false,"M":true}
message sent to kafka {"e":"trade","E":1712386965492,"s":"BTCUSDT","t":3534946849,"p":"68100.00000000","q":"0.07341000","b":26291532617,"a":26291518302,"T":1712386965491,"m":false,"M":true}
message sent to kafka {"e":"trade","E":1712386967211,"s":"BTCUSDT","t":3534946850,"p":"68100.00000000","q":"0.07341000","b":26291532773,"a":26291518302,"T":1712386967211,"m":false,"M":true}
message sent to kafka {"e":"trade","E":1712386967430,"s":"BTCUSDT","t":3534946851,"p":"68099.99000000","q":"0.00398000","b":26291532342,"a":26291532803,"T":1712386967429,"m":true,"M":true}
message sent to kafka {"e":"trade","E":1712386967494,"s":"BTCUSDT","t":3534946852,"p":"68100.00000000","q":"0.00101000","b":26291532813,"a":26291518302,"T":1712386967493,"m":false,"M":true}
message sent to kafka {"e":"trade","E":1712386967848,"s":"BTCUSDT","t":3534946853,"p":"68100.00000000","q":"0.00148000","b":26291532848,"a":26291518302,"T":1712386967847,"m":false,"M":true}
message sent to kafka {"e":"trade","E":1712386968205,"s":"BTCUSDT","t":3534946854,"p":"68100.00000000","q":"0.07341000","b":26291532869,"a":26291518302,"T":1712386968205,"m":false,"M":true}
message sent to kafka {"e":"trade","E":1712386968469,"s":"BTCUSDT","t":3534946855,"p":"68099.99000000","q":"0.00338000","b":26291532342,"a":26291532889,"T":1712386968468,"m":true,"M":true}
message sent to kafka {"e":"trade","E":1712386968649,"s":"BTCUSDT","t":3534946856,"p":"68099.99000000","q":"0.00094000","b":26291532342,"a":26291532898,"T":1712386968649,"m":true,"M":true}
message sent to kafka {"e":"trade","E":1712386969941,"s":"BTCUSDT","t":3534946857,"p":"68100.00000000","q":"0.00635000","b":26291533184,"a":26291518302,"T":1712386969941,"m":false,"M":true}
message sent to kafka {"e":"trade","E":1712386969941,"s":"BTCUSDT","t":3534946858,"p":"68100.00000000","q":"0.03000000","b":26291533184,"a":26291518313,"T":1712386969941,"m":false,"M":true}
message sent to kafka {"e":"trade","E":1712386969941,"s":"BTCUSDT","t":3534946859,"p":"68100.00000000","q":"0.03706000","b":26291533184,"a":26291518414,"T":1712386969941,"m":false,"M":true}
message sent to kafka {"e":"trade","E":1712386970047,"s":"BTCUSDT","t":3534946860,"p":"68100.00000000","q":"0.00054000","b":26291533192,"a":26291518414,"T":1712386970046,"m":false,"M":true}
message sent to kafka {"e":"trade","E":1712386970062,"s":"BTCUSDT","t":3534946861,"p":"68100.00000000","q":"0.00078000","b":26291533196,"a":26291518414,"T":1712386970061,"m":false,"M":true}
message sent to kafka {"e":"trade","E":1712386970353,"s":"BTCUSDT","t":3534946862,"p":"68100.00000000","q":"0.00073000","b":26291533211,"a":26291518414,"T":1712386970352,"m":false,"M":true}





PART 2

TASK 1

objective: Determine the percentage of trades where the buyer is the market maker for each trading pair.

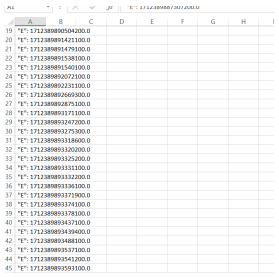
We are using HIVE to execute the following

```
-- t104_109_118_132 - task 1
```

```
/* Create a table to store the data from the csv file that we
generated in part 1.
We will also define the schema here
*/

 CREATE EXTERNAL TABLE btcusdt_trades
 (
    trade_id INT,
    trading_pair STRING,
    buyer STRING,
    seller STRING,
    price DECIMAL(10, 2),
    quantity DECIMAL(10, 2),
    trade_timestamp TIMESTAMP
)
/*
This command gives us some information as to how the values a
making it easier to store in our table
*/
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/root/btcusdt_trades';

/*In the next commad we select the fiels required and perform
aggragate function ( calculating market maker trades and perc
of market makers trades)and also mentions where the ouput fil
*/
INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/task1.csv'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
SELECT trading_pair, COUNT(*) AS total_trades,
    SUM(CASE WHEN buyer = 'Market Maker' THEN 1 ELSE 0 END) AS
    ROUND((SUM(CASE WHEN buyer = 'Market Maker' THEN 1 ELSE 0
FROM btcusdt_trades
GROUP BY trading_pair;
```

The output is a csv file whose contains look like this

```
"E": 1712389887307,2,0,0.0
"E": 1712389887327,5,0,0.0
"E": 1712389887390,1,0,0.0
"E": 1712389887539,2,0,0.0
"E": 1712389887794,3,0,0.0
"E": 1712389888011,1,0,0.0
"E": 1712389888763,1,0,0.0
"E": 1712389888841,1,0,0.0
"E": 1712389889102,1,0,0.0
"E": 1712389889111,1,0,0.0
"E": 1712389889490,1,0,0.0
"E": 1712389889538,1,0,0.0
"E": 1712389889565,1,0,0.0
"E": 1712389889671,1,0,0.0
"E": 1712389889688,1,0,0.0
"E": 1712389889878,1,0,0.0
"E": 1712389890000,1,0,0.0
"E": 1712389890272,1,0,0.0
"E": 1712389890504,2,0,0.0
"E": 1712389891421,1,0,0.0
"E": 1712389891479,1,0,0.0
"E": 1712389891538,1,0,0.0
"E": 1712389891540,1,0,0.0
"E": 1712389892072,1,0,0.0
"E": 1712389892231,1,0,0.0
"E": 1712389892669,3,0,0.0
"E": 1712389892875,1,0,0.0
"E": 1712389893171,1,0,0.0
"E": 1712389893247,2,0,0.0
"E": 1712389893275,3,0,0.0
"E": 1712389893318,6,0,0.0
"E": 1712389893320,2,0,0.0
"E": 1712389893325,2,0,0.0
"E": 1712389893331,1,0,0.0
"E": 1712389893332,2,0,0.0
"E": 1712389893336,1,0,0.0
```



```
"E": 1712389893371,9,0,0.0
"E": 1712389893374,1,0,0.0
"E": 1712389893378,1,0,0.0
"E": 1712389893437,1,0,0.0
"E": 1712389893439,4,0,0.0
"E": 1712389893488,1,0,0.0
"E": 1712389893537,1,0,0.0
"E": 1712389893541,2,0,0.0
"E": 1712389893593,1,0,0.0
"E": 1712389893645,1,0,0.0
"E": 1712389893699,1,0,0.0
"E": 1712389893757,1,0,0.0
"E": 1712389893879,2,0,0.0
"E": 1712389894147,11,0,0.0
"E": 1712389894149,1,0,0.0
"E": 1712389929432,15,0,0.0
"E": 1712389929490,2,0,0.0
"E": 1712389929555,1,0,0.0
"E": 1712389929616,1,0,0.0
"E": 1712389929688,2,0,0.0
"E": 1712389929690,15,0,0.0
"E": 1712389929701,2,0,0.0
"E": 1712389929738,2,0,0.0
"E": 1712389929818,1,0,0.0
"E": 1712389929924,1,0,0.0
"E": 1712389929926,3,0,0.0
"E": 1712389930019,2,0,0.0
"E": 1712389930145,4,0,0.0
"E": 1712389930197,1,0,0.0
"E": 1712389930251,1,0,0.0
"E": 1712389930254,1,0,0.0
"E": 1712389930271,1,0,0.0
"E": 1712389930308,5,0,0.0
"E": 1712389930311,28,0,0.0
"E": 1712389930348,1,0,0.0
"E": 1712389930359,2,0,0.0
"E": 1712389930370,1,0,0.0
"E": 1712389930396,1,0,0.0
```

```
"E": 1712389930400,15,0,0.0
"E": 1712389930410,11,0,0.0
"E": 1712389930414,2,0,0.0
"E": 1712389930423,3,0,0.0
"E": 1712389930428,2,0,0.0
"E": 1712389930467,14,0,0.0
"E": 1712389930470,7,0,0.0
```

```
hive> CREATE EXTERNAL TABLE btcusdt_trades (
    >     trade_id INT,
    >     trading_pair STRING,
    >     buyer STRING,
    >     seller STRING,
    >     price DECIMAL(10, 2),
    >     quantity DECIMAL(10, 2),
    >     trade_timestamp TIMESTAMP
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE
    > LOCATION '/user/root/btcusdt_trades';
OK
Time taken: 0.666 seconds
```

```
hive> INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/task1.csv'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > SELECT
    >     trading_pair,
    >     COUNT(*) AS total_trades,
    >     SUM(CASE WHEN buyer = 'Market Maker' THEN 1 ELSE 0 END) AS market_maker_trades,
    >     ROUND((SUM(CASE WHEN buyer = 'Market Maker' THEN 1 ELSE 0 END) / COUNT(*)) * 100, 2) AS percentage_market_maker_trades
    > FROM
    >     btcusdt_trades
    > GROUP BY
    >     trading_pair;
Query ID = root_20240406082908_4b6a1f70-8361-468c-8a86-72a2eee489ea
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1712389738936_0002, Tracking URL = http://18a885818b32:8088/proxy/application_1712389738936_0002/
Kill Command = /usr/local/hadoop/bin/mapred job  -kill job_1712389738936_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2024-04-06 08:29:14,612 Stage-1 map = 0%,  reduce = 0%
2024-04-06 08:29:19,780 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.65 sec
2024-04-06 08:29:24,955 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.61 sec
MapReduce Total cumulative CPU time: 6 seconds 610 msec
Ended Job = job_1712389738936_0002
Moving data to local directory /home/hadoop/task1.csv
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.61 sec   HDFS Read: 63440 HDFS Write: 2275 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 610 msec
OK
Time taken: 18.999 seconds
```

TASK 2

Objective: Find the top 5 trading pairs with the highest trade volume.

We are going to use MapReduce to solve this question.

We are using the csv file that was generated in part 1. There are 2 parts -

1. mapper code

   This code reads the input, then parses JSON object and extracts necessary fields.

   The output → a pair and volume

```
#!/usr/bin/env python3
import json
import sys

for line in sys.stdin:
    tevent = json.loads(line.strip())
    pair = (tevent['b'], tevent['a'])
    vol= float(tevent['p']) * float(tevent['q'])
    print(pair, "\t", vol)
```

2. reducer code

   Processes the pair (key) and volume (value) generated in the code above. It then sorts the pairs by volume in descending order.

   Output → top 5 pairs along with their volume

```
import sys
from collections import defaultdict as dd
pairs = dd(float)
for line in sys.stdin:
    parts = line.strip().split(')')
    pair_str = parts[0] + ')'
    pair = tuple(map(int, pair_str.strip('()').split(',')))
    volume_str = parts[1].strip()
    volume = float(volume_str)
    pairs[pair] += volume
sorted_res = sorted(pairs.items(), key=lambda x: x[1], revers
```

```
for pair, volume in sorted_res[:5]:
    print(pair, "\t", volume)
```

```
2024-04-06 21:38:32,094 INFO mapreduce.Job: Job job_local2109940132_0001 completed successfully
2024-04-06 21:38:32,169 INFO mapreduce.Job: Counters: 36
        File System Counters
                FILE: Number of bytes read=304720
                FILE: Number of bytes written=1635403
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=91810
                HDFS: Number of bytes written=200
                HDFS: Number of read operations=15
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Map input records=243
                Map output records=243
                Map output bytes=10425
                Map output materialized bytes=10917
                Input split bytes=100
                Combine input records=0
                Combine output records=0
                Reduce input groups=243
                Reduce shuffle bytes=10917
                Reduce input records=243
                Reduce output records=5
                Spilled Records=486
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=35
                Total committed heap usage (bytes)=532676608
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=45905
        File Output Format Counters
                Bytes Written=200
2024-04-06 21:38:32,171 INFO streaming.StreamJob: Output directory: /example/T_104_109_118_132
```

```
pes1ug22am132@pes1ug22am132:~/T_104_109_118_132$ hdfs dfs -cat /example/T_104_109_118_132/part-00000
(26291956216, 26291956696)      9999.5212519
(26291967915, 26291717560)      8370.5943
(26291967936, 26291717560)      8302.4243
(26291967894, 26291717560)      7945.8952
(26291967468, 26290767555)      7464.250365
```