Big Data Processing

Ethereum Analysis Coursework

ECS765P

MSc Big Data Science with Machine Learning

Student: Alex Santonastaso

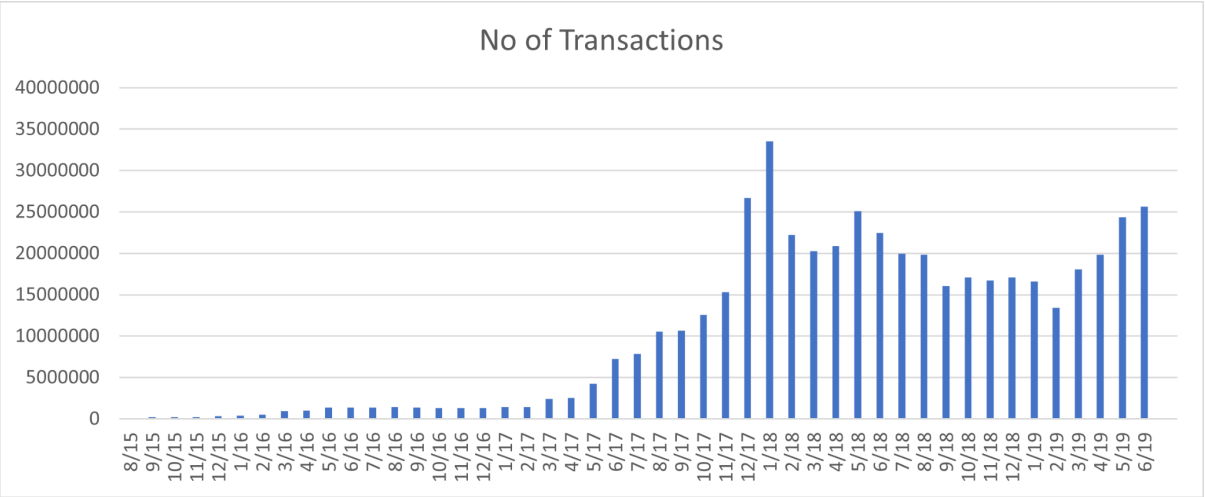
ID: 210991068

Summary

Part A

1. Create a bar plot showing the number of transactions occurring every month between the start and end of the dataset.

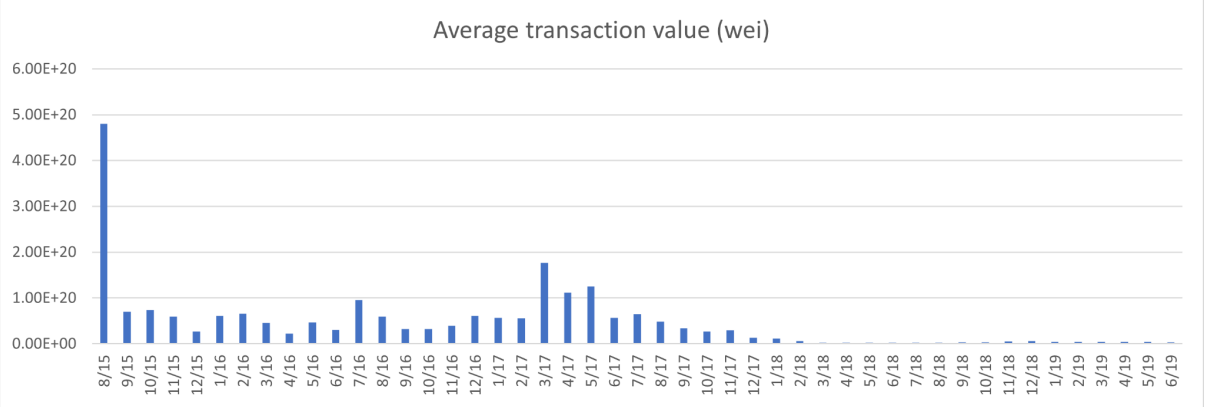
In order to execute this task I ran a Hadoop job consisting of a mapper which firstly splits the dataset at every ‘,’. A try and catch statement checks that the length of each sample in the dataset has a length of 7 values and ignores the rest. Secondly the timestamp is converted to a date consisting of month and year, then date(key) and 1 (value) are yielded. The reducer then counted the number of transactions occurring every month. I then imported the output into an excel spreadsheet, sorted the values by date and plotted the following chart.



Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_22197/

2. Create a bar plot showing the average value of transactions in each month between the start and end of the dataset.

In order to execute this task I ran a Hadoop job consisting of a mapper which firstly splits the dataset at every ‘,’ as we did during the previous one. Again a try and catch statement is used to check that the length of each sample is 7 and ignores the rest. Secondly the timestamp is converted to a date consisting of month and year, then date(key) and transaction value in Wei(value) are yielded. A combiner was then used to count the number of transactions and their sum which are lastly yielded in the reducer after calculating the average. I then imported the output into an excel spreadsheet, sorted the values by date and plotted the following chart.



Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_23950/

Part B - Top ten most popular services

In order to execute this task I am running a hadoop job consisting of two mappers and two reducers: in the first mapper we check if the input samples are from the transactions dataset or from the contracts dataset using length. If the sample is a transaction the address to which the transaction is sent is yielded as key, flag 1 is yielded with the value of the transaction. If the sample is a contract the address is yielded as key, while '(2,1)' is yielded as key, where flag 2 is only used to distinguish between transactions and contracts in the next step. In the first reducer I iterate through the values yielded in the first mapper: if the value is from the transaction dataset ('1'), its value is appended to a list containing all the transactions amounts; if the value is from the contracts dataset('2'), we yield again the addresses as key but this time the value will be the sum of all the transactions amounts. The second mapper yields both the key and value from the first reducer together as value; the second reducer sorts the address with the sums of transactions in descending order, yielding only the first 10 sorted values. Here is the output:

"0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444" 84155100809965865822726776
"0xfa52274dd61e1643d2205169732f29114bc240b3" 45787484483189352986478805
"0x7727e5113d1d161373623e5f49fd568b4f543a9e" 45620624001350712557268573
"0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef" 43170356092262468919298969
"0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8" 27068921582019542499882877
"0xbfc39b6f805a9e40e77291aff27aee3c96915bdd" 21104195138093660050000000
"0xe94b04a0fed112f3664e45adb2b8915693dd5ff3" 15562398956802112254719409
"0xbb9bc244d798123fde783fcc1c72d3bb8c189413" 11983608729202893846818681
"0xabbb6bebfa05aa13e908eaa492bd7a8343760477" 11706457177940895521770404
"0x341e790174e3a4d35b65fdc067b6b5634a61caea" 83790007519177556240575000

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_5980/
Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_6053/

Part C - Evaluate the top 10 miners by the size of the blocks mined.

First mapper splits the blocks dataset at every ‘,’. A try and catch statement is used to check that the length of each sample is 9 and ignores the rest. The mapper then yields every miner wallet address and the corresponding block size, which are nextly aggregated into the first reducer. The second mapper yields a list containing both the miner address and the sum of his mined block sizes as value, which is then sorted in the second reducer using the python “sorted” function. Here is the output:

"0xea674fdde714fd979de3edf0f56aa9716b898ec8"	23989401188
"0x829bd824b016326a401d083b33d092293333a830"	15010222714
"0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c"	13978859941
"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5"	10998145387
"0xb2930b35844a230f00e51431acae96fe543a0347"	7842595276
"0x2a65aca4d5fc5b5c859090a6c34d164135398226"	3628875680
"0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01"	1221833144
"0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb"	1152472379
"0x1e9939daaad6924ad004c2560e90804164900341"	1080301927
"0x61c808d82a3ac53231750dadc13c777b59310bd9"	692942577

Job ID:http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_22452/

Part D

Popular Scams

In this problem we are correlating the count of how many scams for each category are active/inactive/offline/online/suspended with their volumes.In order to solve this problem I am running two hadoop programs. First one gives as output the category and status of scams with their count. First mapper yields address values from transactions dataset with flag 1; category, addresses and status of scam from scams dataset with flag 2. First reducer yields each category and status found with value 1; second reducer yields each category and status with the sum of all its counts. As we can see from the output the most scams found in the dataset are of category ‘Scamming’ and are active.

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:19888/jobhistory/job/job_1648683650522_5337

["Active", "Scamming"] 88444
["Inactive", "Phishing"] 22
["Offline", "Fake ICO"] 121
["Offline", "Phishing"] 7022
["Offline", "Scam"] 0
["Suspended", "Phishing"] 11
["Active", "Phishing"] 1584
["Offline", "Scamming"] 24692
["Suspended", "Scamming"] 56

Second job gives as output the category and status of scam with their volumes. As we can see from the output the most lucrative scams are of category ‘Phishing’ however its status is currently offline, while the most lucrative active are of category ‘Scamming’. The most lucrative by volume are offline phishing scams indeed, as their volume is higher than active scamming one, even though their count is less than 1/10 of active scamming ones(7022 vs 88444).

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:19888/jobhistory/job/job_1648683650522_5377

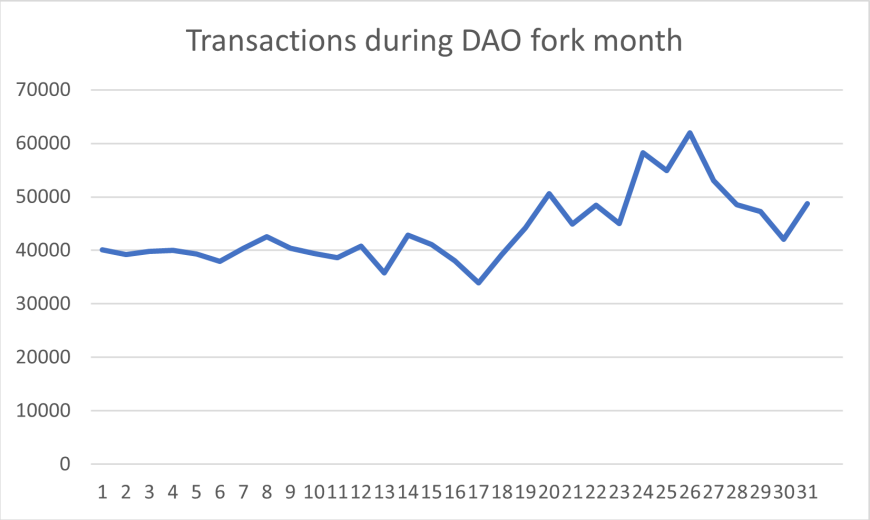
["Active", "Scamming"] 2.20969523566791e+22
["Inactive", "Phishing"] 1.488677770799503e+19
["Offline", "Fake ICO"] 1.35645756688963e+21
["Offline", "Phishing"] 2.2451251236751494e+22
["Offline", "Scam"] 0
["Suspended", "Phishing"] 1.63990813e+18

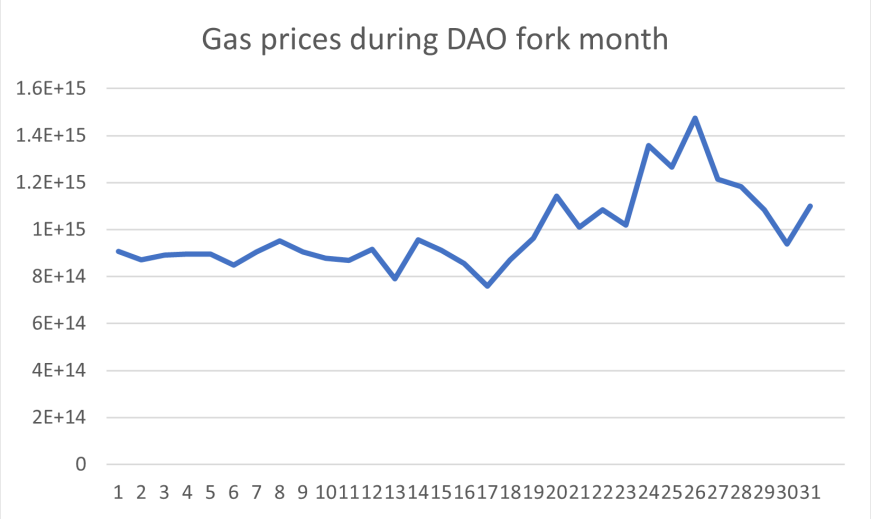
["Active", "Phishing"] 4.5315978714979395e+21
["Offline", "Scamming"] 1.6235500337815102e+22
["Suspended", "Scamming"] 3.71016795e+18

Fork the chain - Analysis of transactions and gas price following DAO hack fork (Ethereum blockchain splits into ETC and ETH)

For this task I ran a job similar to part A, which counts the number of transactions and this time also the average gas price. However, this time we are only looking for these values during the month of July of 2016, as there was a fork on the 16 of this month after the DAO hack, which resulted in the ethereum blockchain to be splitted in two: ETH and ETC. After running the jobs I plotted the outputs in excel:

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_3680





As we can see from the above plots, the fork resulted in an increase in general usage of the Ethereum blockchain. An upper trend is visible following the date of the 16 of april, which was the date the fork happened.

Comparative evaluation - spark

For this task part B problem is solved using spark instead of hadoop framework. I used again transactions and contracts dataset however I imported pyspark instead of MRJob. After loading the datasets I checked that their samples length is correct (7 for transactions, 5 for contracts) using two functions I defined, while the remaining inconsistent samples are ignored. After that I map address and value as key and value, I aggregate all the transaction values for each address. Lastly I map the address as key for contracts, that then I join with the output given from the transaction reducer. The data is then sorted and the top 10 is given as output in a .txt file. I also imported time in order to count how long the spark job takes to run every time and I append this duration in seconds together with the top10 output txt file. Here is the output for the first job run:

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_3684
0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444:84155100809965865822726776
0xfa52274dd61e1643d2205169732f29114bc240b3:45787484483189352986478805
0x7727e5113d1d161373623e5f49fd568b4f543a9e:45620624001350712557268573
0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef:43170356092262468919298969
0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8:27068921582019542499882877
0xbfc39b6f805a9e40e77291aff27aee3c96915bdd:21104195138093660050000000
0xe94b04a0fed112f3664e45adb2b8915693dd5ff3:15562398956802112254719409
0xbb9bc244d798123fde783fcc1c72d3bb8c189413:11983608729202893846818681
0xabbb6bebfa05aa13e908eaa492bd7a8343760477:11706457177940895521770404
0x341e790174e3a4d35b65fdc067b6b5634a61caea:83790007519177556240575 00
191.375396013 seconds

Where this last line is the duration in seconds the spark jobs took to run. I then ran the job 4 more times, and it took the following amount of time.

Second run duration: 119.915675163 seconds
Third run duration: 268.557856798 seconds
Fourth run duration: 296.337949991 seconds
Fifth run duration: 193.31732583 seconds

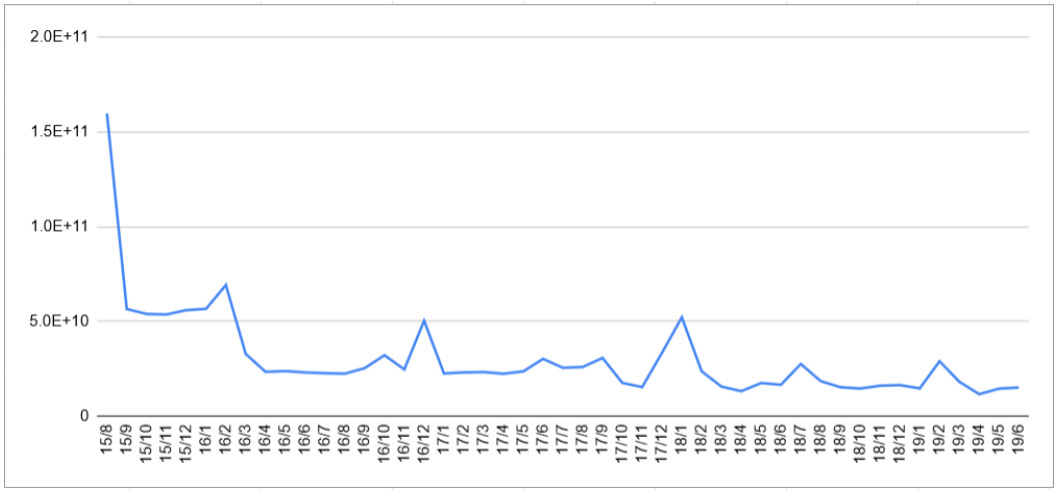
Average spark run length: 214 seconds

On the other hand, the same task solved using hadoop framework took 30 minutes on average in three runs (30m, 28m, 29m approximately), hence ten times longer than spark. Therefore, spark seems more appropriate than hadoop for this task, as it gets the same output in a much shorter amount of time.

Gas Guzzlers

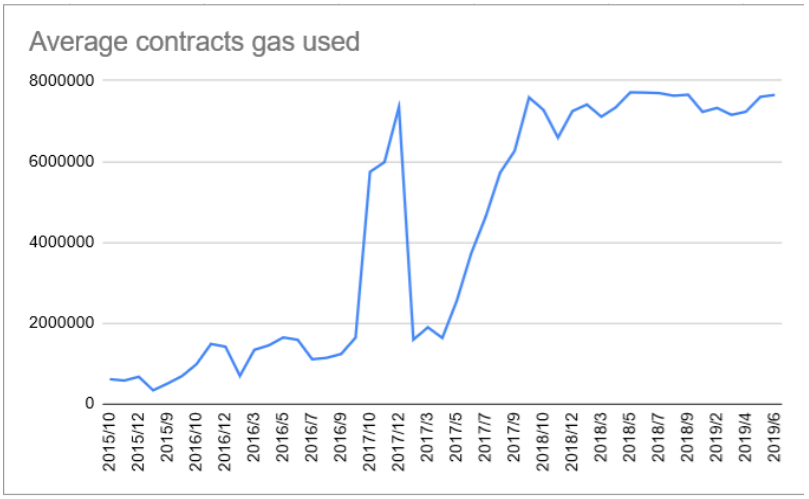
For gas guzzlers analysis I ran three hadoop programs.
First one gives as output the average gas price for each month in the dataset. I then imported the output into an excel spreadsheet, sorted the values by date and plotted the following chart. Gas prices went down over time, probably due to the price of ethereum going up.

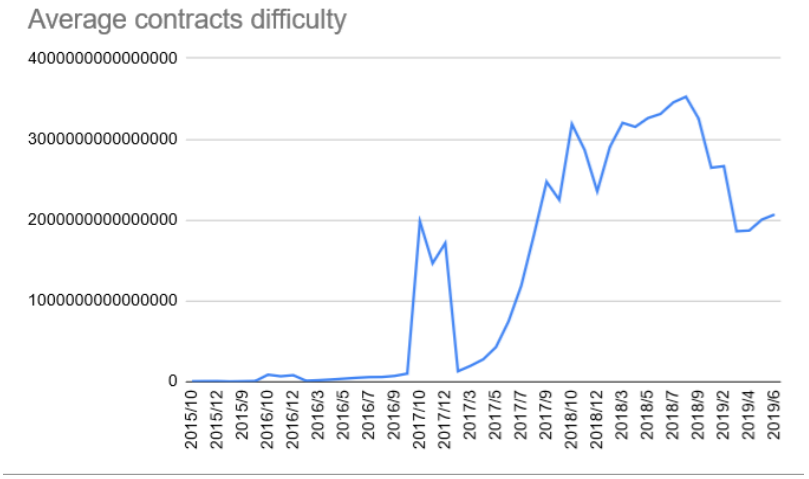
Job ID:http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_3588/



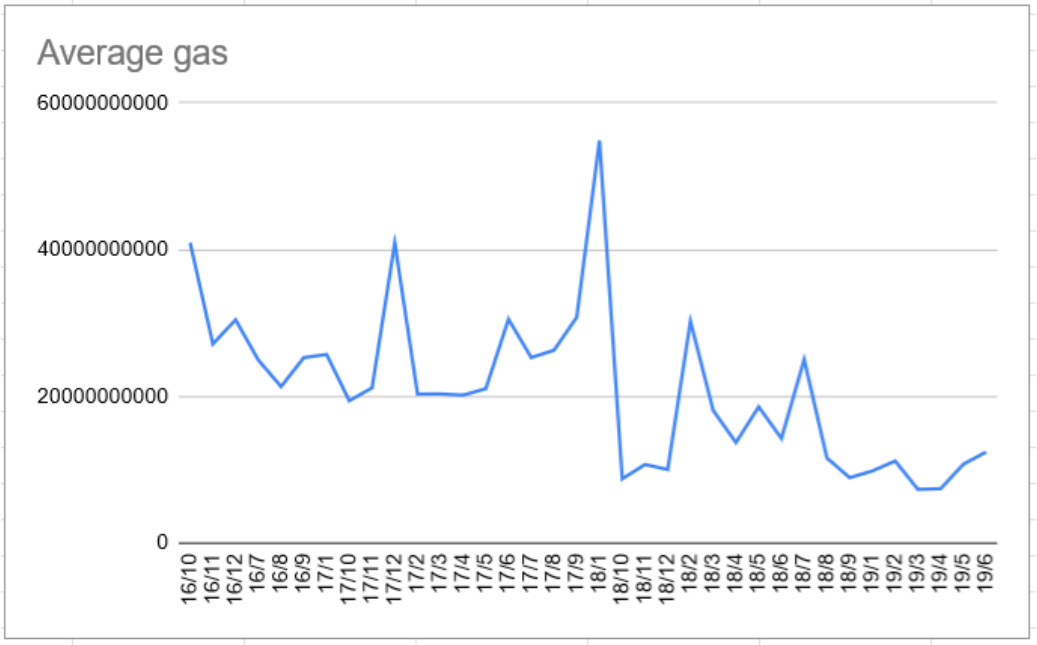
Second one gives as output average gas used and difficulty of each contract for each month in the dataset. I then imported the output into an excel spreadsheet, sorted the values by date and plotted the following chart. Contracts have become more complicated requiring more gas.

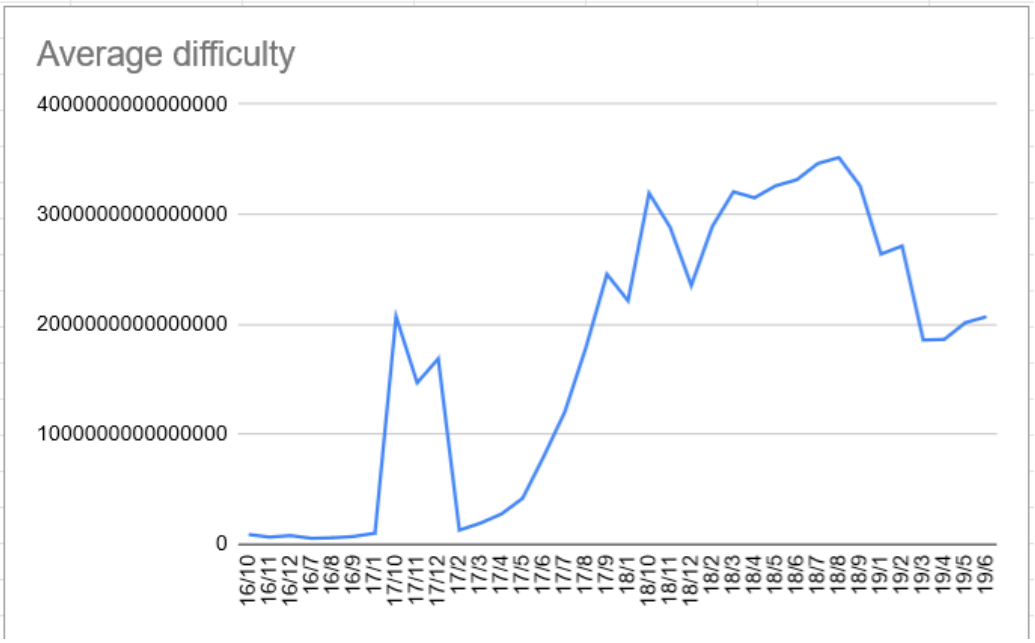
Job ID:http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_3581/





Third one gives as output average gas price and difficulty of top two smart contracts from partB. I then imported the output into an excel spreadsheet, sorted the values by date and plotted the following chart. As we can see from the graph, the average gas price is going down over time, similar to what we saw in the first plot of this problem.





Job ID:http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_3568/