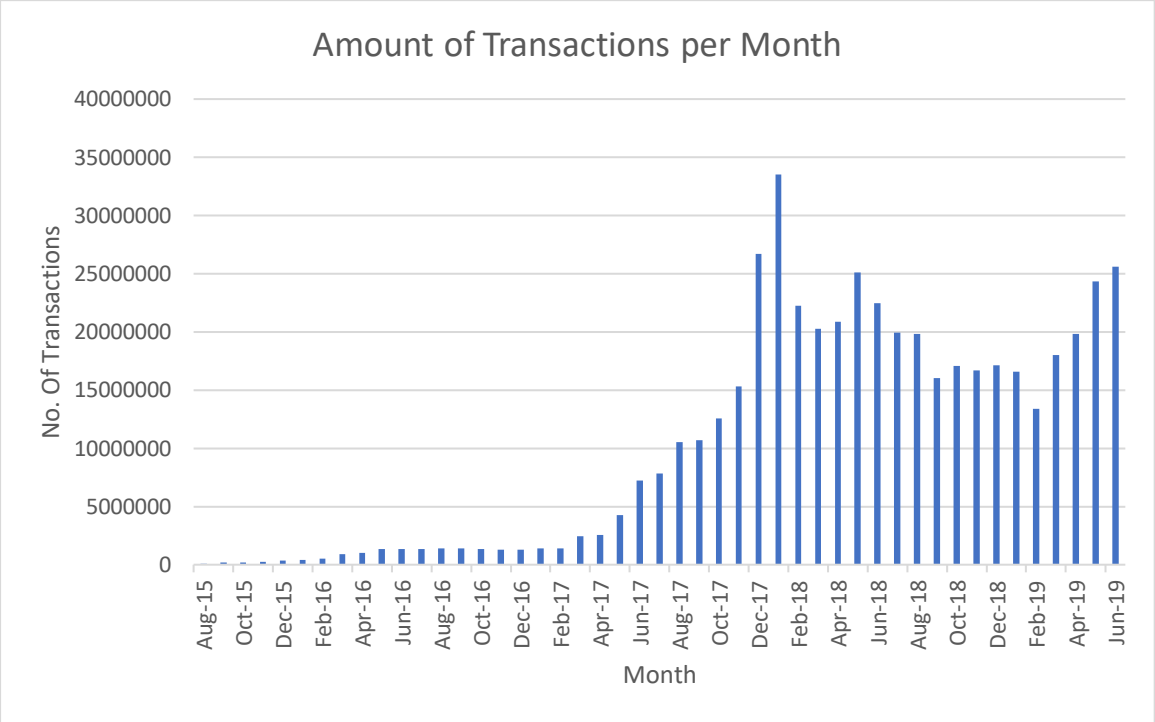
Big Data Coursework

Part 1

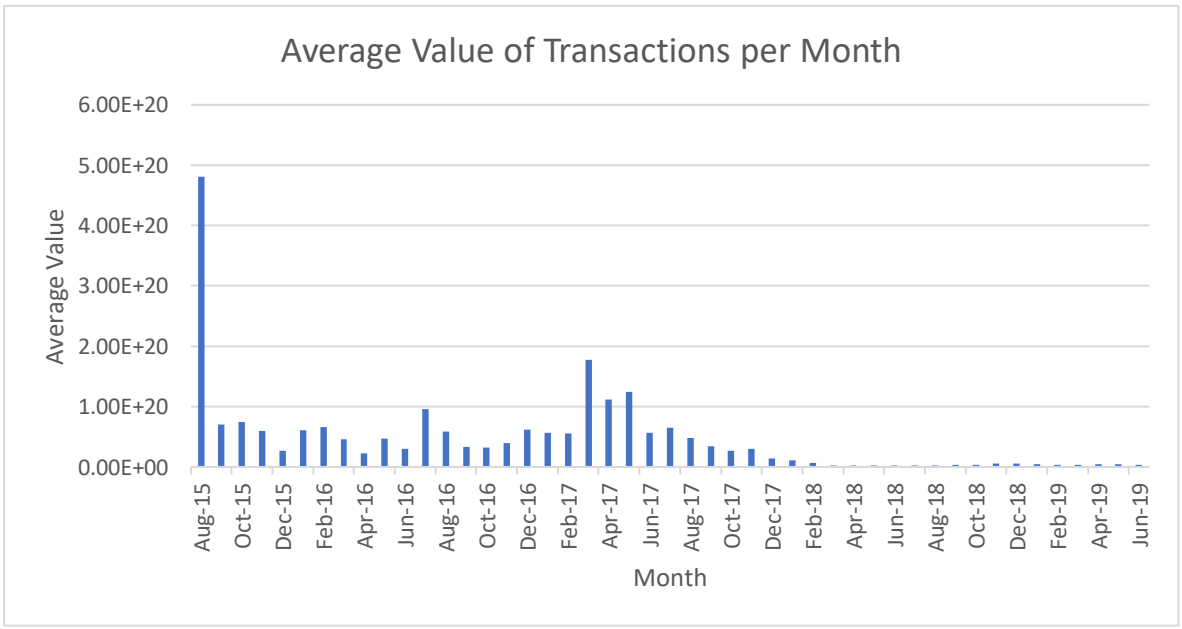
URLS for jobs:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1604349877093_1419/
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1604349877093_1445/
Code for the first part: transPerMonth.py

This is a simple program that takes the date of a transaction and counts the total for a given month and year.

Output after putting data in Microsoft Excel:



Output for second part (code will follow to save space). Excel was also used here:



Code for second part: avgValTrans.py

This code is similar to the one before where the mapper gets the date in a given month and year but it also yields the value and a 1 so that these numbers can be used in the combiner and reducer to calculate the average,

Part B

URLS for jobs:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1604349877093_4038/
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1604349877093_4054/

Code for part B: PartB.py

Explanation:

This one doc of code first uses a mapper to separate the transactions and contract data given to it and yield relevant information depending where the data came from. The reducer then takes all the transactions and only yields them if they are present in the contracts table and if they have a value greater than 0. The next job sorts these to give the top ten most popular servicers.

Output of the file (output was “data” None and was edited to be presented as so) :

0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444 - 841551008099658658227267 76
0xfa52274dd61e1643d2205169732f29114bc240b3 - 45787484483189352986478805
0x7727e5113d1d161373623e5f49fd568b4f543a9e - 45620624001350712557268573
0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef - 43170356092262468919298969
0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8 - 27068921582019542499882877
0xbfc39b6f805a9e40e77291aff27aee3c96915bdd - 21104195138093660050000000
0xe94b04a0fed112f3664e45adb2b8915693dd5ff3 - 15562398956802112254719409
0xbb9bc244d798123fde783fcc1c72d3bb8c189413 - 11983608729202893846818681
0xabbb6bebfa05aa13e908eaa492bd7a8343760477 - 11706457177940895521770404
0x341e790174e3a4d35b65fdc067b6b5634a61caea - 8379000751917755624057500

Part C

URLS for jobs:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1604349877093_6356/
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1604349877093_6358/

code for this part: PartC.py

The code above is like the one in part b but there is no join with the contracts table. This returns the top 10 miners from the transactions table directly.

Output of the file (output was “data” None and was edited to be presented as so) :

0xea674fdde714fd979de3edf0f56aa9716b898ec8 - 23989401188

0x829bd824b016326a401d083b33d092293333a830 - 15010222714

0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c - 13978859941

0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5 - 10998145387

0xb2930b35844a230f00e51431acae96fe543a0347 - 7842595276

0x2a65aca4d5fc5b5c859090a6c34d164135398226 - 3628875680

0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01 - 1221833144

0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb - 1152472379

0x1e9939daaad6924ad004c2560e90804164900341 - 1080301927

0x61c808d82a3ac53231750dadc13c777b59310bd9 - 692942577

Part D

Scams:

URL's:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_7389/
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_7437/

The first part of the this task was to use the code by Dr Joseph Doyle that convers the scams.json to a .csv file. This file was then used with the following code to produce the top 21 scams that have earned the most value and thus are the most lucrative. (process.py)

Main Code: Scam.py

The code here first looks at if the data is coming from the csv file or transaction table. It then does a similar job as the code in part B where only transactions that have an to_address to one that is known to be a scam are yielded. This is then sorted to the top 21 in the next job.

Output:

"0x311f71389e3de68f7b2097ad02c6ad7b2dde4c71 - 16709083588073530571339 " 0
"0x1d60606d8a09b5015d773a80b0c660bb8d91809c - 2067750892013526763661 " 1
"0xba83e9ce38b10522e3d6061a12779b7526839eda - 1990757131520806579663 " 2
"0xef57aa4b57587600fc209f345fe0a2687bc26985 - 1831741671481489993883 " 3
"0x858457daa7e087ad74cdeeceab8419079bc2ca03 - 1630577419133089537315 " 4
"0xaea3846e14bec1ba8f562844d39b1215a1d588c6 - 1283928004373722729601 " 5
"0x4a0d27a1044dd871a93275de5109e5f5efc4d46e - 127905539734237472723 " 6
"0xf0a924661b0263e5ce12756d07f45b8668c53380 - 1213887393874270060501 " 7
"0x3fcb2d173389b7cd8079ef8b439dbd92e7e0ae28 - 1163904128277001401314 " 8
"0x2268751eafc860781074d25f4bd10ded480310b9 - 1151303025790917525566 " 9
"0x3681828da105fc3c44e212f6c3dc51a0a5a6f5c6 - 1086092910876542255474 " 10
"0xed2d26eed06ecfbfd796d1ec551d2a649a31e576 - 894456149695775852360 " 11
"0x4bf722014e54aeab05fcf1519e6e4c0c3f742e43 - 882710017471720973607 " 12
"0x89c98cc6d9917b615257e5704e83906402f0f91f - 799198984446024595162 " 13
"0xf33068d5e798f6519349ce32669d1ec940db1193 - 783982947175108080428 " 14
"0xfbf6f29c126382cf15795bb209ee506a174cc709 - 683708100000000000000 " 15
"0x05dd62c007cde143b402fa5da3937c40c70b4b14 - 625993880562573479200 " 16
"0x536a6ba0d913d5d6a4ce2c6eb7ed0de3c0f0b89e - 611382037721775256849 " 17
"0x1e80dad60d19fb8159af3f440a8ceaa0e5581847 - 608864310392049734534 " 18
"0x474057adf42f9f955e86aa1142740f9d7763e41e - 563587313193105101199 " 19
"0x33f74739ce6be3b2c38af76f5adb3866cb4784c4 - 501593735318758887946 " 20

This information does not provide any information with how scams changed over time. Therefore, the code provided to us in the email was tweaked to the following: process2.py

This instead produces the id and type of scam that id was. With this, we can figure out what the most lucrative type of scam was.

Urls:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_6703/
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_6704/

The following code was used to count all the types of scams and rank them from most used to least used: Scam2.py

Output:

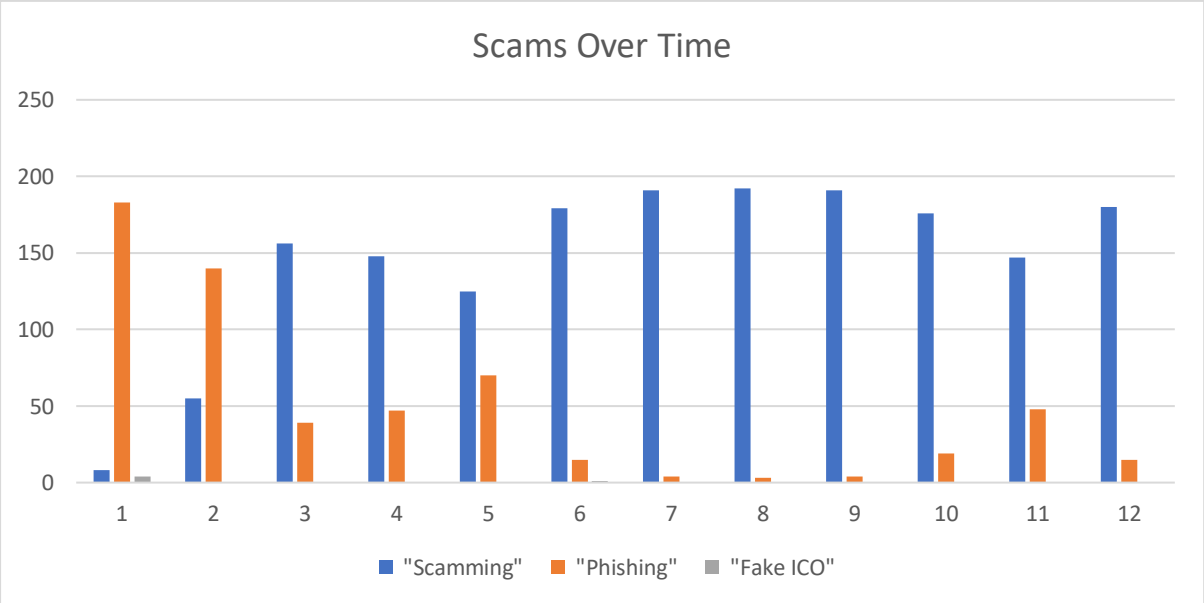
"Scamming - 1747 " 0
"Phishing - 587 " 1
"Fake ICO - 5 " 2
"Scam - 1 " 3

It is clearly visible that type “Scamming” is the most lucrative by a big gap. In order to see the effect of these scams over time, I made the assumption that their id corresponds to a sequential order in which they were reported. This means that we can order these scams with their id and look at their effects over time.

Due to this fact, I divided the data into 12 bins with 195 pieces of data each and used the following script to separate them: reOrder.py

URL’s:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_6827/
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_6825/

I then got the data and put it into excel, with a little bit of clean up, I got this graph:



It is worthy to note that I merged the single “scam” data item with “Scamming”.

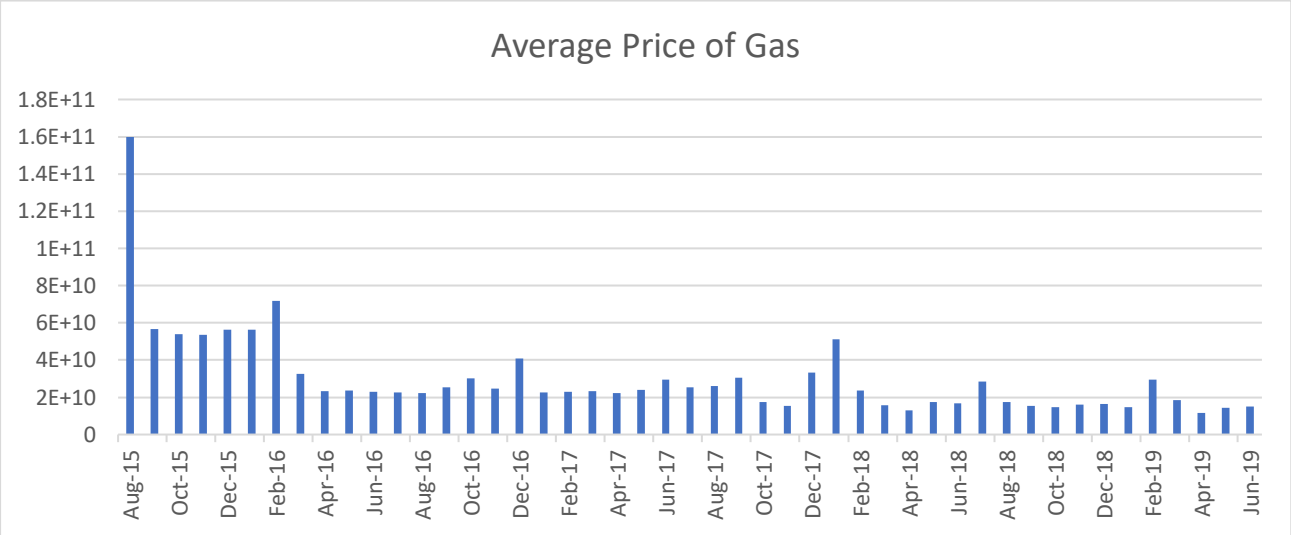
The graph clearly shows that phishing started off being the most popular type of scam but then was quickly replaced by other types of “Scamming”. Once “Scamming” became popular, it has stayed that way and it is the most lucrative type.

Gas Guzzlers:

URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_7537/

Code File: Gas.py

This code takes the gas_price and the timestamp of that transaction at that price and averages them out per month for a given year. This is similar to what was done in part A.

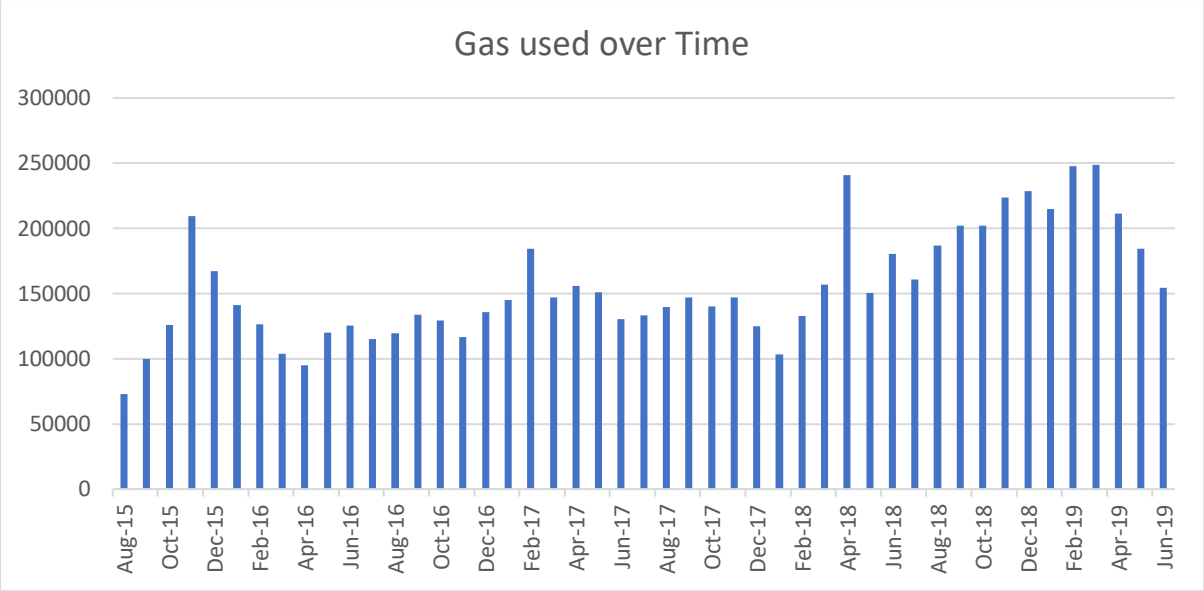


The output was then put into excel and then this graph above was created. This shows that the price of gas started off being high but then levelled out from April of 2016 onwards. To answer the next part of the question, we need to see how much gas was used with the same time unit.

Code File: Ammount.py

URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_7592/

The code on the previous page does the same thing as the code before but it averages the gas used over time. After putting the data into excel, the following chart was created:

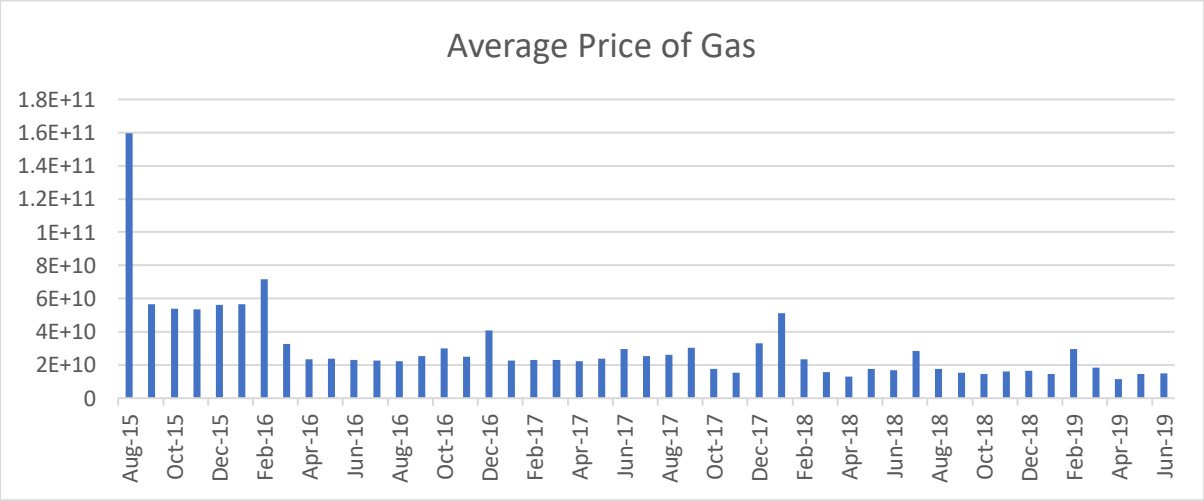


Here we can see that the trend is the opposite to the price. Over time more gas is being used on average. This means that jobs are becoming more complicated to compensate, the price has stayed steady but on a slight decrease.

Fork the Chain:

For this task, there was no code ran, instead, the results of previous tasks can be inspected for the analysis.

In “Gas Guzzlers”, the first graph is a good place to start for this analysis:

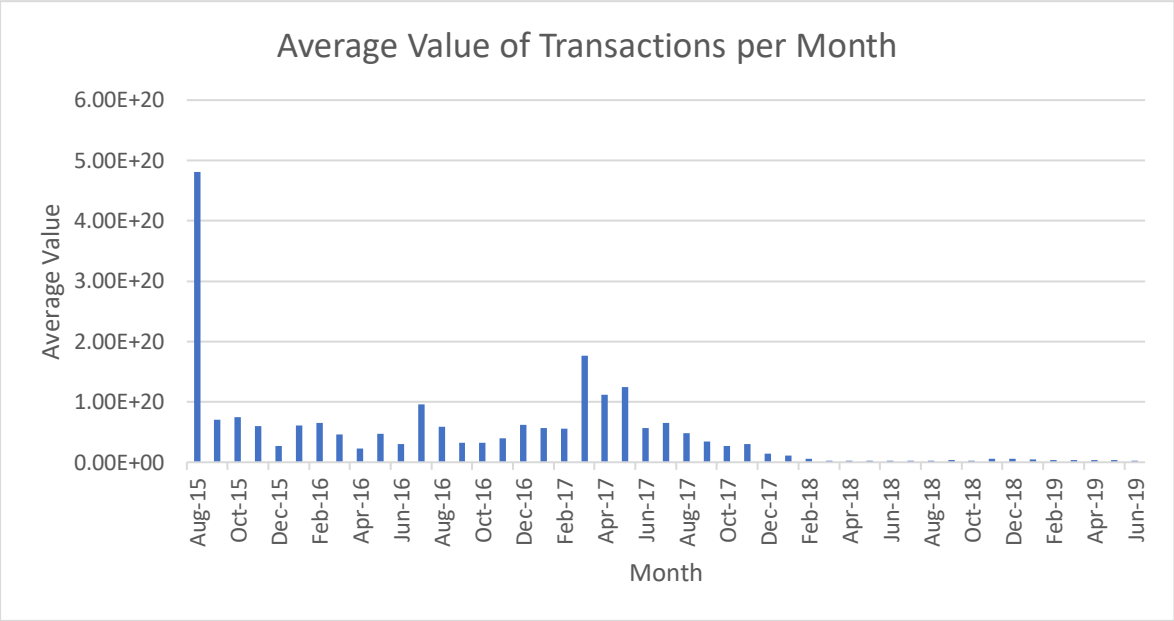


We can inspect this graph for changes that may correlate to forks for Ethereum.

The first major jump of average gas prices can be seen in September of 2015. This Jump also correlates to the fork that is referred to “Ice Age” that was released on “08-09-2015”. This change caused the price of Ethereum to drop quite significantly.

The next jump in price can be seen in the graph to take place from February of 2016 to March of that year. This also correlates to a fork called “Homestead” that took place on March 15th of that year.

Other forks that I could find didn’t have a significant effect on the price of gas, nor did it have a noticeable effect on the usage of gas.



The average value of transaction per moth supports the first fork that I found (This graph is taken from part A). This is less so for the second fork I identified earlier.

To see who made the most money from these forks, I ran two jobs. Each job looked at one of the forks described above and the same script was tweaked to run both. The differences are commented on the actual script.

Main Code: MoneyMade.py

The script sums up all the addresses that made the most money between those moths listed above. So, for the first fork discussed, this was for all transactions between August 2015 and September 2015.

URL’s:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_1166/
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_1172/

These are the results:
"\"\" - 11903261927002084163205159 " 1
"0x32be343b94f860124dc4fee278fdcbd38c102d88 - 6892439892104840013575168 " 2
"0x2910543af39aba0cd09dbb2d50200b3e800a63d2 - 6765111756174844106772707 " 3
"0xb794f5ea0ba39494ce839613fffba74279579268 - 3925099001191659842715648 " 4
"0xe28e72fcf78647adce1f1252f240bbfaebd63bcc - 1131161197472032142729942 " 5
"0x120a270bbc009644e35f0bb6ab13f95b8199c4ad - 557366445307732846661000 " 6

"0xddfafdbc7c90f1320e54b98f374617fbd01d109f - 480687722000000000000000 " 7
"0x510e222df10b146f813acc5b94cbb2a9d1a47ade - 416668100000000000000000 " 8
"0xd268fb48fa174088a25a120aff0fd8eb0c2d4c87 - 396008100000000006291456 " 9
"0xc47aaa860008be6f65b58c6c6e02a84e666efe31 - 378422406036826808009783 " 10
"0x19ae27e1b61445eb9f27821a5eb4b3e957319e7e - 346507264999999998981365 " 11
"0x6000b846630e4f8ae883a270c118f681bd12e593 - 323203436000000000000000 " 12
"0x65d7e41bc79587db2febbcea0ad9d76469c88085 - 323203436000000000000000 " 13
"0x119058dc2c577e9c4ba6914678aa9db565300ffe - 323203436000000000000000 " 14
"0xd30ecf8897af28bccf7e05250ea9e1e275ee41d4 - 241035119000000000000000 " 15
"0xfbb1b73c4f0bda4f67dca266ce6ef42f520fbb98 - 231002460503394227982377 " 16
"0x40b9b889a21ff1534d018d71dc406122ebcf3f5a - 219028875072294902915119 " 17
"0x355f44e5dee45414dd2502d1abf99aa641f8d4d2 - 215468957000000000000000 " 18
"0x2a869ce3f74335f5bdeffb9f3a30d3887721060f - 215001000000000000000000 " 19
"0x314c7daa4825ffc0293819119c5fdcc0ccf19827 - 212300980000000000000000 " 20

It is unclear why the first address is as such but other that that one, the top 19 are displayed underneath.

These are the results from the second job with the dates changed to look for transactions between February 2016 and March 2016:

URL's:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_1190/
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_1194/

"0x32be343b94f860124dc4fee278fdcbd38c102d88 - 132108112169975999239618 56 " 1
"0x2910543af39aba0cd09dbb2d50200b3e800a63d2 - 506906323549773297234676 7 " 2
"0xb794f5ea0ba39494ce839613fffba74279579268 - 460000000000000000251658 24 " 3
"0x120a270bbc009644e35f0bb6ab13f95b8199c4ad - 249830735011340583361802 4 " 4
"0x0c5437b0b6906321cca17af681d59baf60afe7d6 - 238091210899543140285581 9 " 5
"0xa2942dc76c4085295b7a6064a1cfa4d93c18d9bb - 172535390000000000000000 " 6
"0x19ae27e1b61445eb9f27821a5eb4b3e957319e7e - 129709991000017000000000 0 " 7
"0x7180eb39a6264938fdb3effd7341c4727c382153 - 112223960677678118213259 6 " 8
"0x60e16961ad6138d2fb3e556fc284d9c2fff41486 - 101320079000000000000000 " 9
"0xd12cd8a37f074e7eafae618c986ff825666198bd - 951376850000789998417920 " 10
"0xfbb1b73c4f0bda4f67dca266ce6ef42f520fbb98 - 798799542053356560276300 " 11
"0xcafb10ee663f465f9d10588ac44ed20ed608c11e - 784640116541529991000000 " 12
"0x0e10335fe21e84d8ade2708f1f84c1caf4fc498c - 587942190000000001048576 " 13
"0x7a94a7474302b5a508c8e3aa4bea786d436c62bc - 571028250000000000000000 " 14
"0x40b9b889a21ff1534d018d71dc406122ebcf3f5a - 473630416734370000000000 " 15
"0xd94c9ff168dc6aebf9b6cc86deff54f3fb0afc33 - 469948300000000000000000 " 16
"0xf0160428a8552ac9bb7e050d90eeade4ddd52843 - 462972153903513187949575 " 17
"0xf42ac567772ceb9089de2b091d2aedcd78c4c88e - 445439448380000000000000 " 18
"0xc7fe0b8a8bae060c88811f921543d8698192610d - 332999010000000000000000 " 19
"0x8b0cdf6ebdb32412bd1bdc5591502c564122f8b7 - 302946000101009995871232 " 20

Comparative Evaluation:

ID Taken from Hadoop: application_1607539937312_2637

Spark Python file: (Renamed hence it is different to what is on Hadoop) SparkPartB.py

Explanation:

The code uses two methods to see if a line is valid for transactions and contracts. The rest of the code cleans the data using these methods and then goes to join them, sum them, and then sort the data.

(u'0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444', 84155100809965865822726776L)
(u'0xfa52274dd61e1643d2205169732f29114bc240b3', 45787484483189352986478805L)
(u'0x7727e5113d1d161373623e5f49fd568b4f543a9e', 45620624001350712557268573L)
(u'0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef', 43170356092262468919298969L)
(u'0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8', 27068921582019542499882877L)
(u'0xbfc39b6f805a9e40e77291aff27aee3c96915bdd', 21104195138093660050000000L)
(u'0xe94b04a0fed112f3664e45adb2b8915693dd5ff3', 15562398956802112254719409L)
(u'0xbb9bc244d798123fde783fcc1c72d3bb8c189413', 11983608729202893846818681L)
(u'0xabbb6bebfa05aa13e908eaa492bd7a8343760477', 11706457177940895521770404L)
(u'0x341e790174e3a4d35b65fdc067b6b5634a61caea', 83790007519177556240575O0L)
The Spark python code has given the same output as the map reduce job from part B. The first observation and thus comparison is that this job executed significantly faster than the map reduce job.

I used python's time module to time how long the spark job took 5 different times and averaged them out:

T1: 158.516259909
T2: 117.821849108
T3: 159.261208057
T4: 130.431666851
T5: 165.856976032

Average:
146.377592

Therefore, on average, this spark job takes 2.4 minutes to complete the job whereas the map reduce job takes well over 10 minutes at times. I was not able to get an accurate time for the map-reduce job as Hadoop was not working and not accepting the same code as used in part B. But when it did run for part B, it took over 10 minutes.

Therefore, Spark is better for this type of job. The main reason for this is because Hadoop needs to run two different maps reduce jobs that can be far more time consuming. Spark on the other hand will have the RDD's ready in memory to use and operate and hence is faster. If a job only required one map-reduce task, Hadoop is arguably better and maybe faster, but Spark is more ideal otherwise.