Ethereum Analysis

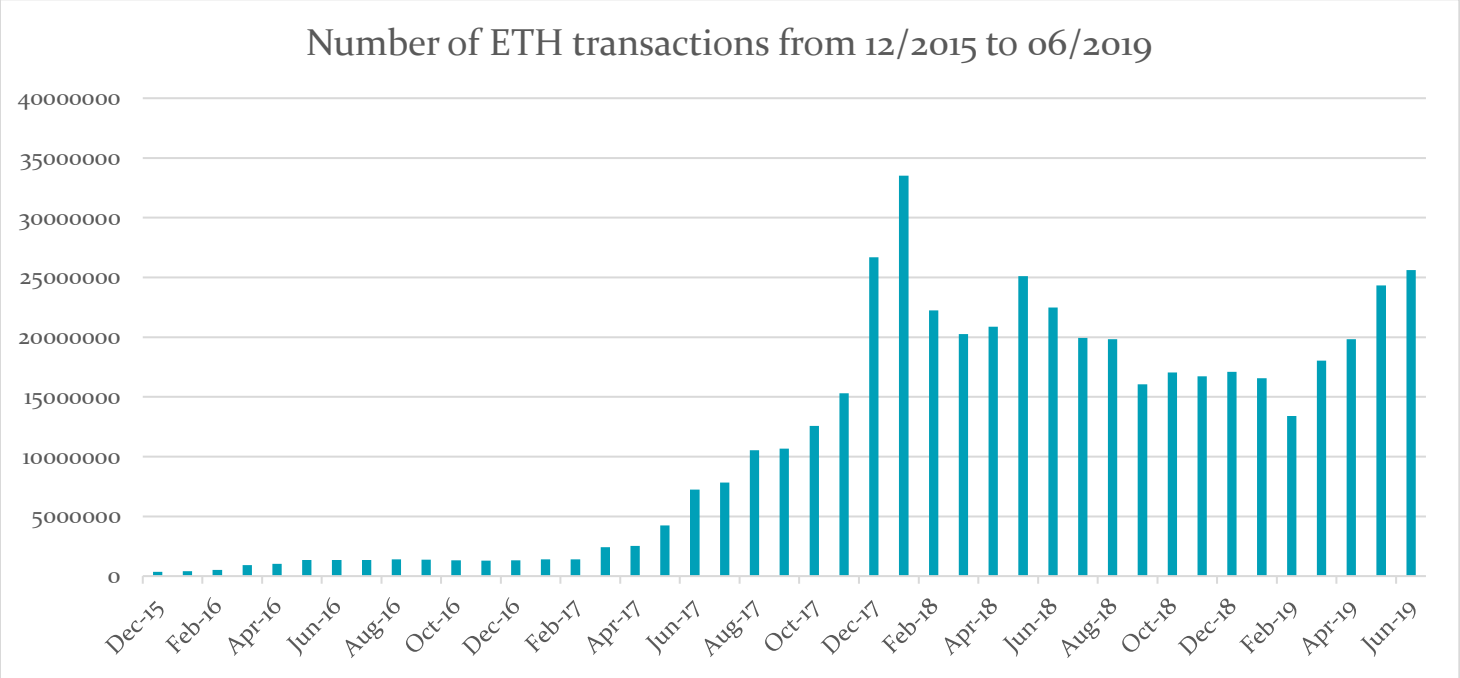
BIG DATA PROCESSING

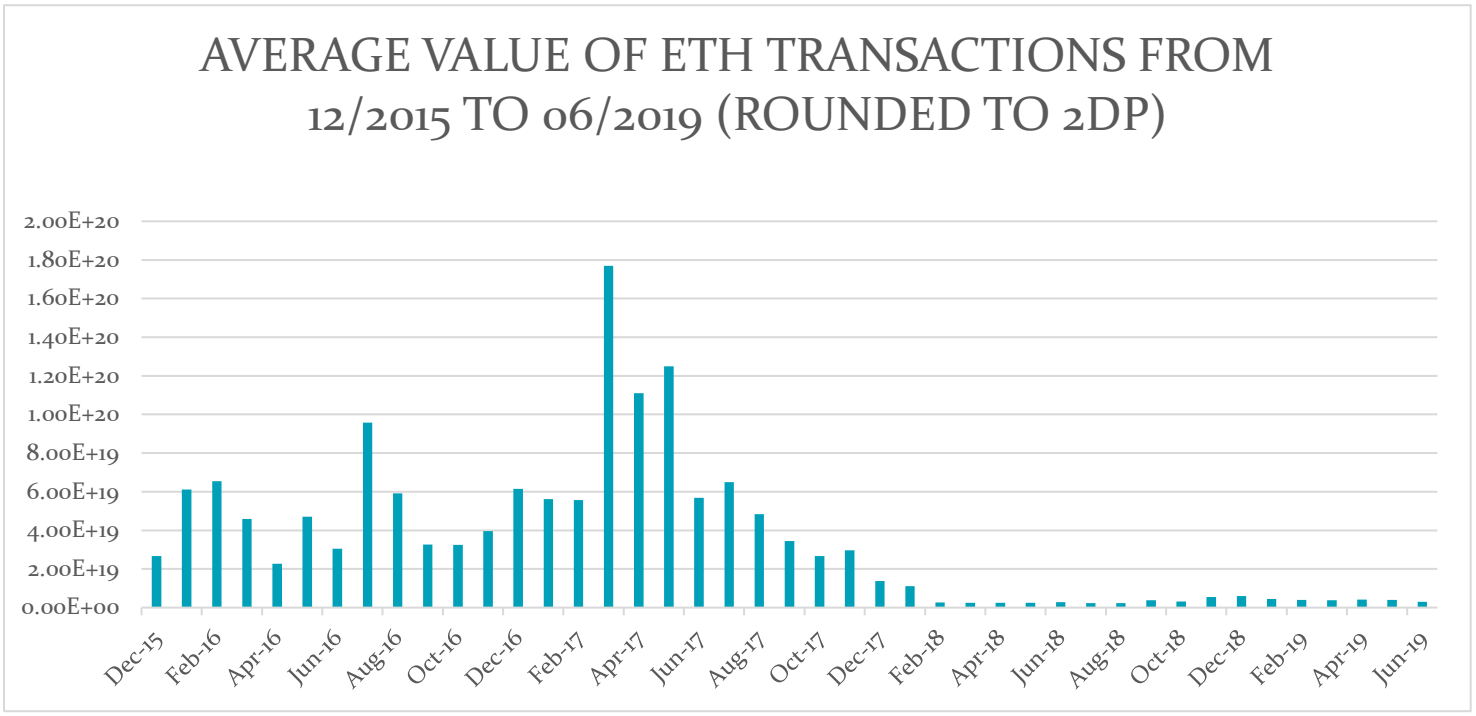
Bishes Limbu | | 190217781

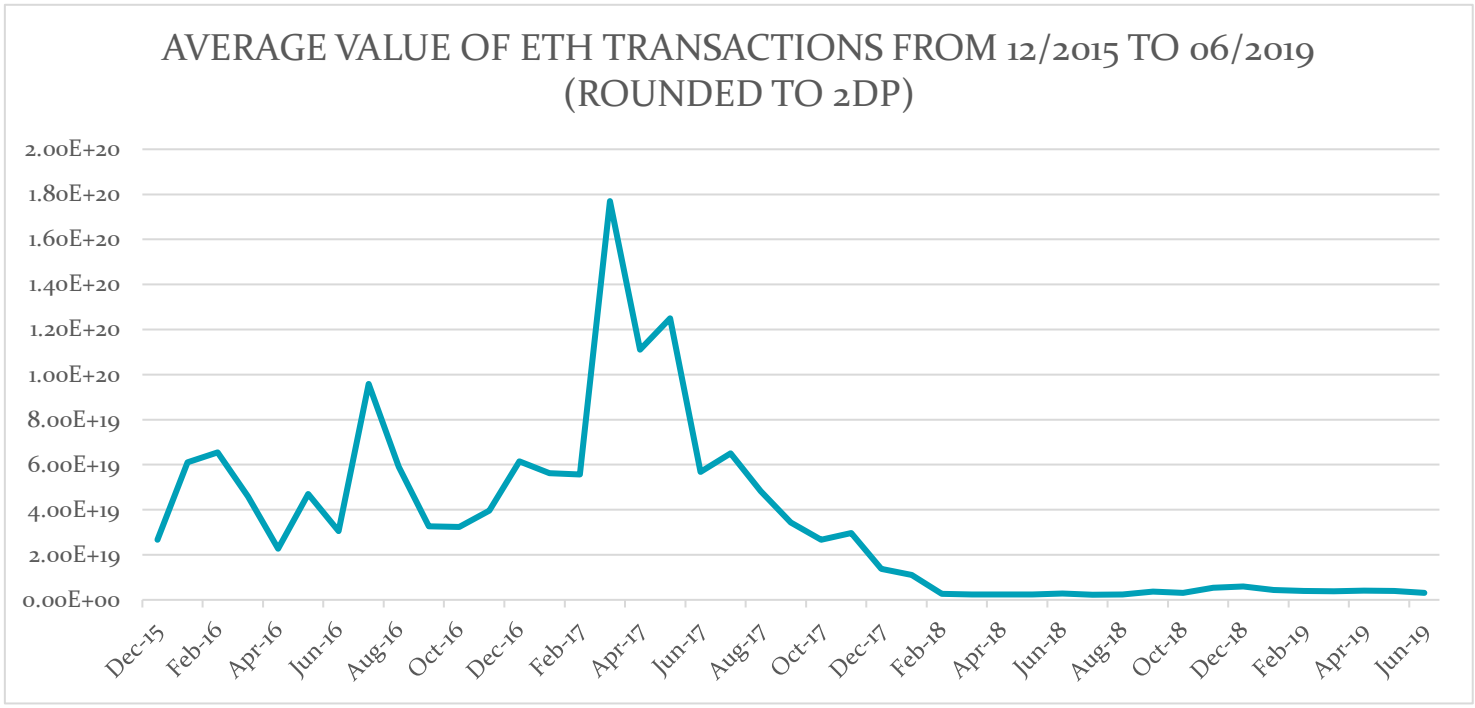
PART A

- Mapper took in lines from the CSV transactions Hadoop dataset. It used the data set to extract the timestamp, which was then converted to a readable month and year. The month and year were put into a new variable called date to represent month and year. This was passed/yielded to the reducer as a Key and 1 was passed the value, which represented 1 transaction per line. In the reducer we total up the number of transactions in each month of each year, then we output this.

- The mapper code/logic is the same as before, however we add a new addition. We also retrieve the value from the transaction dataset and we also change the yield of the mapper. The date is still the key, however we now pass a tuple (value,1). In the reducer we calculate the average of the values for each month + year and output the month year and average value.




AVERAGE VALUE OF ETH TRANSACTIONS FROM 12/2015 TO 06/2019 (ROUNDED TO 2DP)
2.00E+20
1.80E+20
1.60E+20
1.40E+20
1.20E+20
1.00E+20
8.00E+19
6.00E+19
4.00E+19
2.00E+19
0.00E+00
Dec-15
Feb-16
Apr-16
Jun-16
Aug-16
Oct-16
Dec-16
Feb-17
Apr-17
Jun-17
Aug-17
Oct-17
Dec-17
Feb-18
Apr-18
Jun-18
Aug-18
Oct-18
Dec-18
Feb-19
Apr-19
Jun-19


AVERAGE VALUE OF ETH TRANSACTIONS FROM 12/2015 TO 06/2019 (ROUNDED TO 2DP)
2.00E+20
1.80E+20
1.60E+20
1.40E+20
1.20E+20
1.00E+20
8.00E+19
6.00E+19
4.00E+19
2.00E+19
0.00E+00
Dec-15
Feb-16
Apr-16
Jun-16
Aug-16
Oct-16
Dec-16
Feb-17
Apr-17
Jun-17
Aug-17
Oct-17
Dec-17
Feb-18
Apr-18
Jun-18
Aug-18
Oct-18
Dec-18
Feb-19
Apr-19
Jun-19

PART B

JOB 1 // INITIAL AGGREGATION

Used the transactions data set to create an aggregation. Using the to_address as the key and the value as the join value. This was passed to the reducer and the value for each to_address was summed into a total. This is then filtered, to remove any 0 values, since we do not need them, then we yield the filtered results, so the text file has the to_addresses and the totaled values for each address.

JOB 2 // JOINING AGGREGATION TO CONTRACTS

Using the initial aggregation file, we used the to address to join the dataset with the contracts dataset. In the mapper we used if statements to check which dataset the line belonged coming to the mapper belonged to, using the length of the fields that the dataset was split into.

For example since the contracts dataset had 5 fields and our initial aggregation had 2. You can tell which dataset if came from. From the initial aggregation, we use the to_address as a the join key and the value as the join value. While in contracts, we just use to_address as a join key while yielding 0 as the join value.

Both ifs, will yield the join key with a 1 or a 2, this is so that in the reducer we can differentiate which dataset the yield from the mapper came from.

In the reducer, we use a Boolean (which is set to false at the start of the reducer) to check if the current to_address coming from the mapper exists in both files. We use a for loop to loop and if statements to check which dataset the yield belong too. However, using the 1 or 2 in the tuple mentioned before we can do this easily.

If the reducer receives a line from the initial aggregation it sums up the value for that key (to address). Then it checks again if that key exists in the contracts data set. If it does then it will set the Boolean to true.

There is then a final if outside the loop that checks filters using the Boolean if the key exists in both datasets and if the totaled value is bigger than 0. It then yields the key and the value.

JOB 3 // TOP 10 OF JOINED DATA.

Mapper takes the joined dataset and extracts the address and the value. Yields the 2 in a tuple as the value, while using None as a key to send it all to the same reducer.

In the reducer, we use a for loop, array and a rank variable. In the for loop we append all the tuples we get from the mapper into an unsorted array. Outside the for loop, we then use the python sorted function to sort all the tuples in the array by the 2nd value in the tuple, (which is the value number). Afterwards we use store sortedarray into top10 by using [:10] to split the array into the first 10 array elements, so we only get the top10 values.

Afterwards we loop through this new top10 sorted array and output the values and to addresses and rank, and we increment the rank variable every loop.

```
1       "0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444" || 8415510080996585772784025 6"
2       "0xfa52274dd61e1643d2205169732f29114bc240b3" || 4578748448318935627569561 6"
3       "0x7727e5113d1d161373623e5f49fd568b4f543a9e" || 4562062400135071359041536 0"
4       "0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef" || 4317035609226246599422771 2"
5       "0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8" || 2706892158201954237729996 8"
6       "0xbfc39b6f805a9e40e77291aff27aee3c96915bdd" || 2110419513809366044953804 8"
7       "0xe94b04a0fed112f3664e45adb2b8915693dd5ff3" || 1556239895680211203653632 0"
8       "0xbb9bc244d798123fde783fcc1c72d3bb8c189413" || 1198360872920289408502988 8"
9       "0xabbb6bebfa05aa13e908eaa492bd7a8343760477" || 1170645717794089581779353 6"
10      "0x341e790174e3a4d35b65fdc067b6b5634a61caea" || 8379000751917756107980800"
```

PART C

AGGREGATION

Same logic/code as the 1st aggregation, only reimplemented to find the miner and the size.

TOP 10

Same logic/code as the part b job 3, only reimplemented to find the top 10 miners.

```
1       "0xea674fdde714fd979de3edf0f56aa9716b898ec8" || 23989401188"
2       "0x829bd824b016326a401d083b33d092293333a830" || 15010222714"
3       "0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c" || 13978859941"
4       "0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5" || 10998145387"
5       "0xb2930b35844a230f00e51431acae96fe543a0347" || 7842595276"
6       "0x2a65aca4d5fc5b5c859090a6c34d164135398226" || 3628875680"
7       "0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01" || 1221833144"
8       "0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb" || 1152472379"
9       "0x1e9939daaad6924ad004c2560e90804164900341" || 1080301927"
10      "0x61c808d82a3ac53231750dadc13c777b59310bd9" || 692942577"
```

PART D – POPULAR SCAMS

CONVERT JSON TO CSV

Using the JPARSE code given to convert the scams.json file of known scams into a CSV file with the values we need, which include the address and scam id.

SCAMS AGGREGATION

Same logic as p1, aggregate all the scams, however there is an extra addition. Due to the fact that a scam id has multiple addresses. There is a extra if statement in the mapper to check for these extra addresses. We check theses by checking the length of the fields of the SCAM CSV. If it is longer than 5, that means there are extra addresses. This means we need to loop from the position of 4, 1 before the start of the addresses, so that we can yield all the extra addresses as well.

Reducer takes in the values, and yields the key which is the scam id and the extra addresses for each scam id.

SCAM AGGREGATION JOIN TRANSACTIONS

Using the previous logic, and a repartition join to join the scams aggregation to the transactions by using the to_address as the join key.

TOP 10 SCAMS

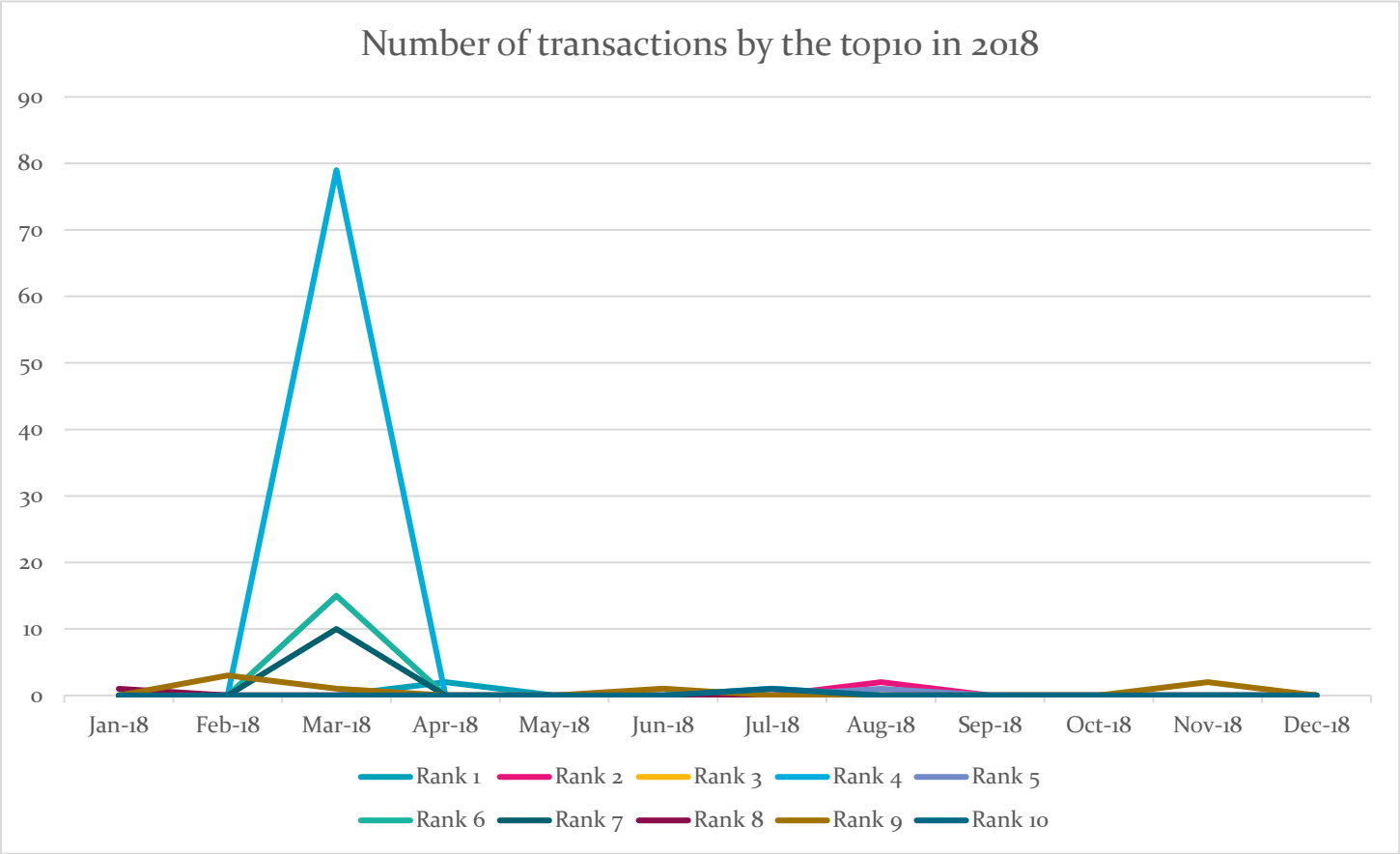
Implemented the same previous code, but to calculate the top 10 scams, with the ids of the scam.

```
1       "3333||0xd243018b2825ad409512a200e744529bc1b129f2||9999.16"
2       "5232||0x8db82c361c101cac757d709200b23ffc8e69fad2||9998.740399999999"
3       "3039||0x6067a95e7bb071a5b73741628a0a5cb5cc164203||998983.612399459"
4       "3065||0x078c2b3b09528fd7b80c9ee715f378d382f9139b||99748.5007"
5       "5983||0xc0ecbcd3a5977921fafecffdf96b4bcae1573b99||9950.0"
6       "4258||0x4f1872383be22878af5d4795b69be61b35ec5d10||995.17"
7       "5447||0x995000b1f9429233a53f53f6475a3efdd8daed11||9940.0"
8       "4172||0xb9b848702a47ae78a70d18c45e552ab1637b5908||9905.5938"
9       "3056||0xaa90ae1b969bc96e82c80e834b5f739e335b7e47||99041.13249999999"
10      "4762||0xb0d46eb7e6aac501bcce47f05ace37527b8b329a||9900.0"
```

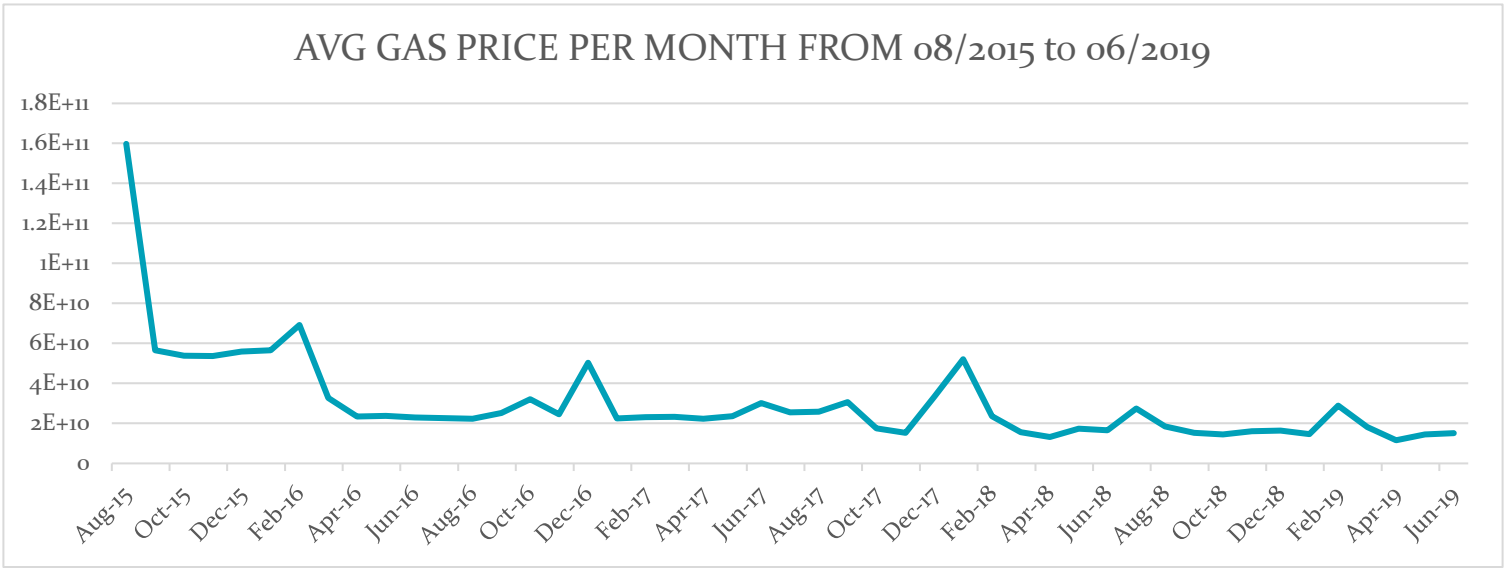
Scam ID after the rank, to address next and the total value.

NUMBER OF TRANSACTIONS OF TOP 10 SCAMS, IN 2018

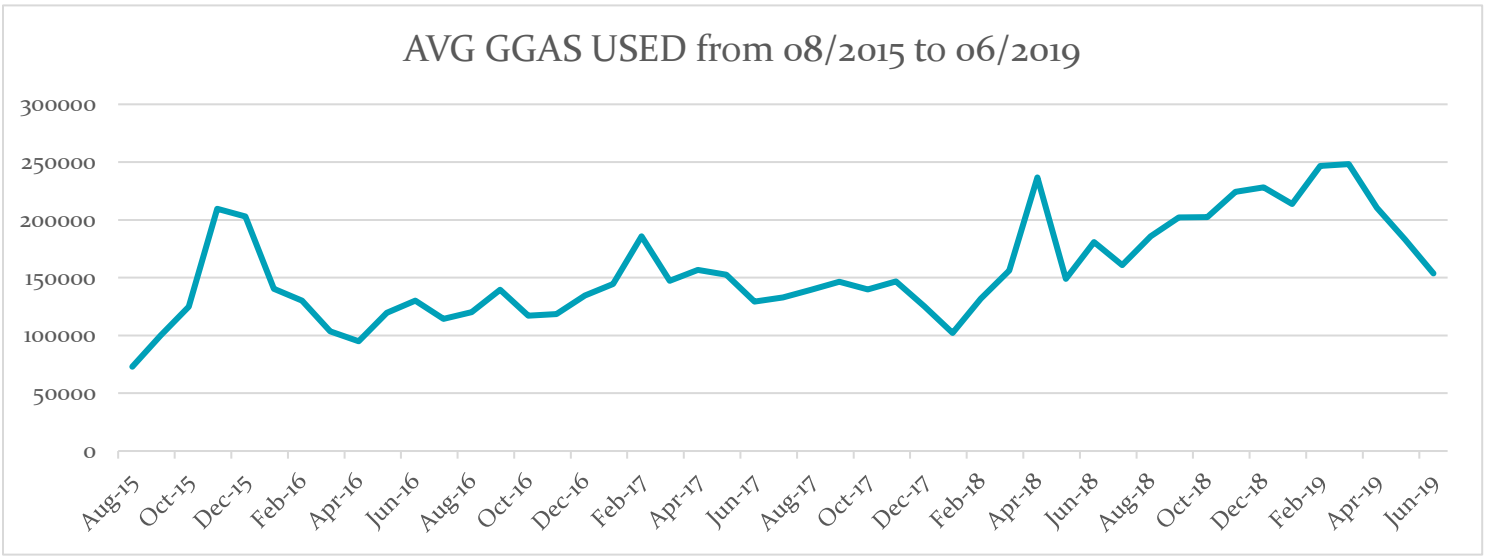
Joining transactions and scams to create a new top10, with all the transaction dates for each scam id in the top 10 scams in 2018. Here we see the top 10 ranking scams and the number of times the scams underwent transactions. We can see that scam id at rank 4, in the top 10 has accumulated 79 transactions in march, while the others have ad a smaller amount of transactions but over the months have more, while the rank 4 scam does not.



PART D – GAS GUZZLERS



Average Gas price has been decreasisng from the start August of 2015.



Here you can see the average of gas used increasing from August 2015

This was done by aggregating all the gas used/prices by using the month and year as a key yielded from the mapper, this was then passed to the reducer to total the values. In the reducer before we yield, we divide the aggregate gas by the number of transactions as well so that we yield the average gas price or average gas used.