Ethereum Analysis

Big Data Processing

Name: Varun Singh

Time Analysis

Here we create two bar plots to show the number of Ethereum transactions occurring per month and the average Ethereum transactions occurring per month.

- The input file we use for this question is: **transactions.csv**
- Code: **q1a.py**
- Command used to execute in terminal:

ccc create spark q1a.py -d -s

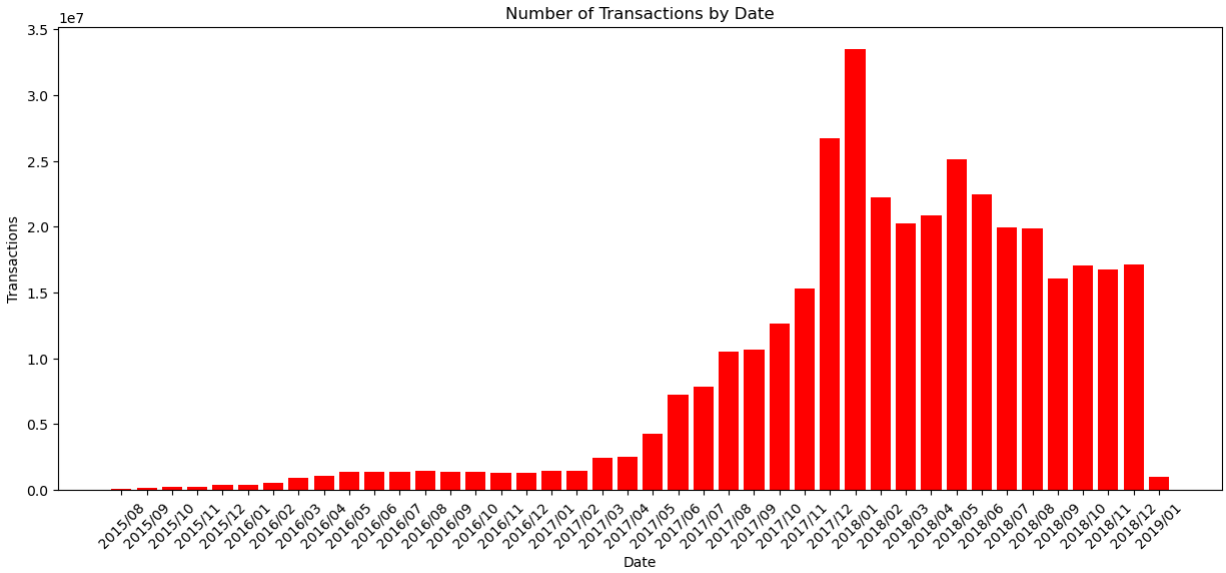
ccc method bucket cp bkt:My_eth_05_04_23_13:48:43/trans_output.txt ~/BDNN_coursework/

Methodology:

To perform the first task, monthly time and transaction count data was obtained . By mapping the date and count from the transaction dataset and applying the reduceByKey function, the monthly transaction count was obtained. This output was saved as "trans_output.txt" and then converted to a CSV file using the code under part 1a in Google Colab's "BDP.ipynb" file to represent monthly transaction totals.

The output data can be found in trans_output.txt

Bar Plot:



- The input file we use for this question is: **transactions.csv**
- Code: **q1b.py**
- Command used to execute in terminal:

ccc create spark q1b.py -d -s

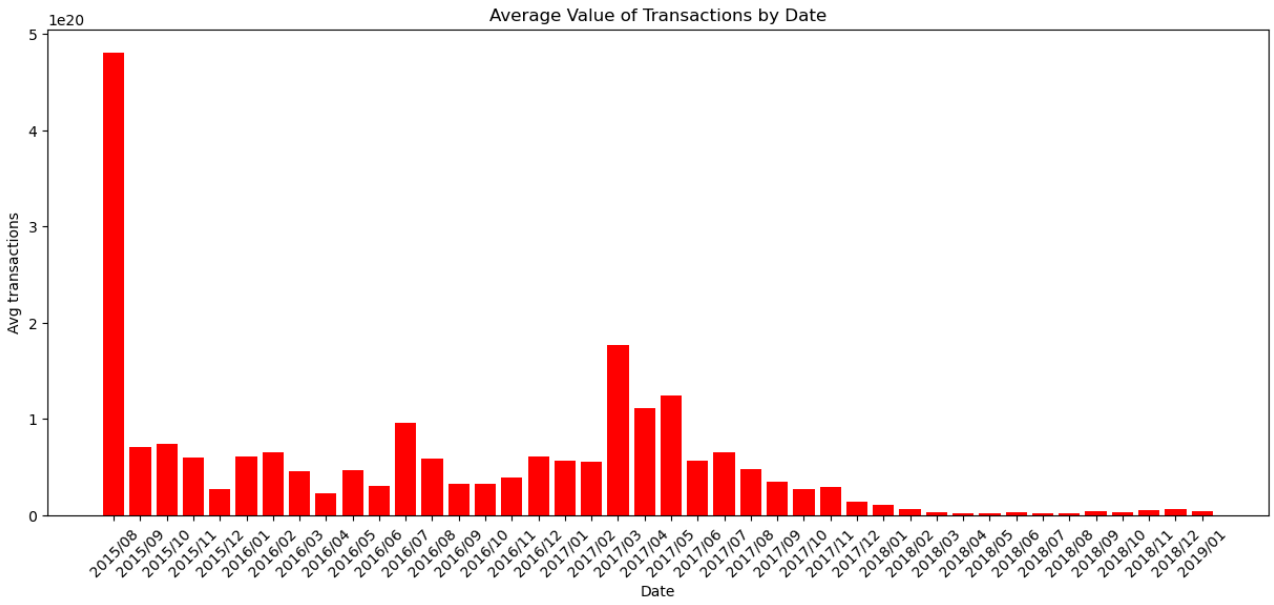
ccc method bucket cp bkt:ethereum_avg05_04_23_14:28:21/avg_trans.txt ~/BDNN_coursework/

Methodology:

To complete the second task, monthly time and average transaction value data was obtained. Utilizing the transaction dataset, the date, value, and count to obtain the values and number of transactions for each month were mapped. These values were then reduced using the reduceByKey function to obtain the total values and total number of transactions for each month. Afterwards, the data was mapped with the date and the total value was divided by the total number of transactions to derive the average transaction value for each month. The output was saved as "avg_trans.txt" and subsequently converted to a CSV file via code under part1b in Google Colab's BDP.ipynb file to represent the average transaction value for each month.

The output data can be found in avg_trans.txt

Bar Plot:



Top Ten Most Popular Services

Here we aim to evaluate the top 10 smart contracts by total Ether received.

- The input file we use for this question is: **transactions.csv contracts.csv**
- Code: **q2.py**
- Command used to execute in terminal:

ccc create spark q2.py -d -s

ccc method bucket cp bkt:eth_/t10_contracts.txt ~/BDNN_coursework/

Methodology:

To initiate this section, we read the contracts and transactions dataset. We map the to_address and value from the transactions dataset, while we map the address and count from the contracts dataset. We join both datasets using .join() with the to_address and address. I then mapped the values and addresses together and identified the top 10 smart contracts using the takeOrdered function. Finally, I saved the output in the text file "t10 _contracts.txt".

Output file obtained: t10_contracts.txt

data:

	id	val
0	0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444	8.42E+25
1	0x7727e5113d1d161373623e5f49fd568b4f543a9e	4.56E+25
2	0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef	4.26E+25
3	0xbfc39b6f805a9e40e77291aff27aee3c96915bdd	2.11E+25
4	0xe94b04a0fed112f3664e45adb2b8915693dd5ff3	1.55E+25
5	0xabbb6bebfa05aa13e908eaa492bd7a8343760477	1.07E+25
6	0x341e790174e3a4d35b65fdc067b6b5634a61caea	8.38E+24
7	0x58ae42a38d6b33a1e31492b60465fa80da595755	2.9E+24
8	0xc7c7f6660102e9a1fee1390df5c76ea5a5572ed3	1.24E+24
9	0xe28e72fcf78647adce1f1252f240bbfaebd63bcc	1.17E+24

Top Ten Most Popular Services

Here we find the top 10 most active miners as per size of block mined.

- The input file we use for this question is: **blocks.csv**
- Code: **q3.py**
- Command used to execute in terminal:

ccc create spark q3.py -d -s

ccc method bucket cp bkt:eth05_04_23_15:52:22/t10_miners.txt ~/BDNN_coursework/

Methodology:

For this section, only the blocks dataset is necessary. We map the size and miner together then reduce them using the reduceByKey function to obtain the block size. We identify the top 10 miners using the takeOrdered function. We save the output in the t10_miners.txt

Output file obtained: t10_miners.txt

data:

	id	miners
0	0xea674fdde714fd979de3edf0f56aa9716b898ec8	1.75E+10
1	0x829bd824b016326a401d083b33d092293333a830	1.23E+10
2	0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c	8.83E+09
3	0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5	8.45E+09
4	0xb2930b35844a230f00e51431acae96fe543a0347	6.61E+09
5	0x2a65aca4d5fc5b5c859090a6c34d164135398226	3.17E+09
6	0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb	1.15E+09
7	0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01	1.13E+09
8	0x1e9939daaad6924ad004c2560e90804164900341	1.08E+09
9	0x61c808d82a3ac53231750dadc13c777b59310bd9	6.93E+08

Data exploration

Scam Analysis

1. **Popular Scams**: Here we aim to find the most lucrative scams and how scams have changed over time.

- The input file we use for this question is: **transactions.csv scams.csv**
- Code: **scams.py**
- Command used to execute in terminal:

ccc create spark scams.py -d -s

ccc method bucket cp bkt:eth_05_04_23_15:52:22/scam_time.txt ~/BDNN_coursework/

ccc method bucket cp bkt:eth_05_04_23_16:22:22/top_scams.txt ~/BDNN_coursework/

Methodology:

The methodology for analyzing the transaction and scam data involves several steps. First, a Spark session is created, environment variables and Hadoop are configured to access the S3 bucket, and the transaction data in CSV format and scam data in CSV format are read from the S3 bucket RDDs are then created from the transaction and scam data and invalid transactions and scams are filtered out. The scams are mapped to their address and ID/category tuples, while the transactions are mapped to their address and Ether received. The datasets are joined on the address and the joined data is mapped to the scam ID/category and Ether received. The total Ether received for each scam ID/category is calculated and the top 15 most lucrative scams are obtained. Similarly, the transactions are mapped to their address and month/year with Ether received, and joined with the scams on the address. The joined data is mapped to the month/year and scam category with Ether received, and the total Ether received for each scam category in each month/year is calculated. The results are uploaded to the S3 bucket and saved with the current date and time as the file name. Finally, the Spark session is stopped.

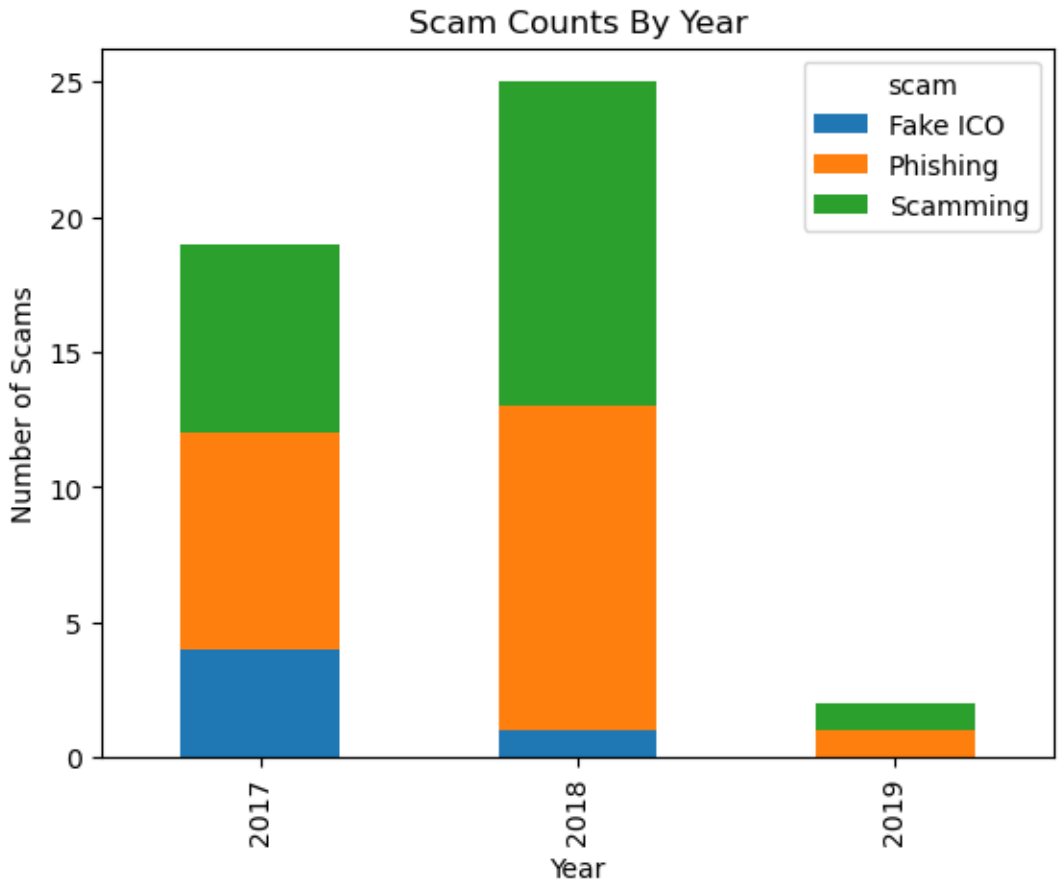
Output file obtained: scam_time.txt & top_scams.txt

Scams over time:

	date	scam	Val
39	2017/05	Phishing	9.00E+16
32	2017/06	Phishing	1.00E+18
8	2017/06	Fake ICO	1.83E+20
25	2017/06	Scamming	9.88E+18
26	2017/07	Fake ICO	1.62E+19
16	2017/07	Phishing	1.06E+22
36	2017/07	Scamming	2.45E+21
40	2017/08	Fake ICO	1.81E+20
11	2017/08	Scamming	3.02E+19
15	2017/08	Phishing	1.42E+22
1	2017/09	Fake ICO	9.75E+20

37	2017/09	Scamming	1.82E+20
10	2017/09	Phishing	3.63E+21
23	2017/10	Phishing	2.77E+21
14	2017/10	Scamming	1.84E+21
35	2017/11	Phishing	3.63E+21
2	2017/11	Scamming	3.31E+18
24	2017/12	Phishing	2.00E+21
41	2017/12	Scamming	2.87E+19
0	2018/01	Scamming	8.03E+20
4	2018/01	Phishing	2.85E+21
13	2018/02	Phishing	5.56E+20
38	2018/02	Scamming	5.48E+20
7	2018/03	Phishing	1.11E+20
3	2018/03	Scamming	3.31E+21
18	2018/04	Phishing	4.31E+20
9	2018/04	Scamming	2.59E+21
5	2018/05	Phishing	9.81E+20
42	2018/05	Scamming	2.07E+21
20	2018/06	Phishing	1.00E+21
21	2018/06	Fake ICO	1.24E+18
19	2018/06	Scamming	2.74E+21
43	2018/07	Scamming	3.39E+21
31	2018/07	Phishing	1.38E+20
34	2018/08	Phishing	4.83E+19
12	2018/08	Scamming	1.19E+21
22	2018/09	Scamming	1.80E+22
6	2018/09	Phishing	3.41E+19
27	2018/10	Scamming	1.76E+21
44	2018/10	Phishing	3.43E+19
28	2018/11	Scamming	2.38E+20
33	2018/11	Phishing	7.53E+19
30	2018/12	Phishing	1.45E+20
17	2018/12	Scamming	4.79E+20
29	2019/01	Scamming	1.53E+18
45	2019/01	Phishing	4.12E+18

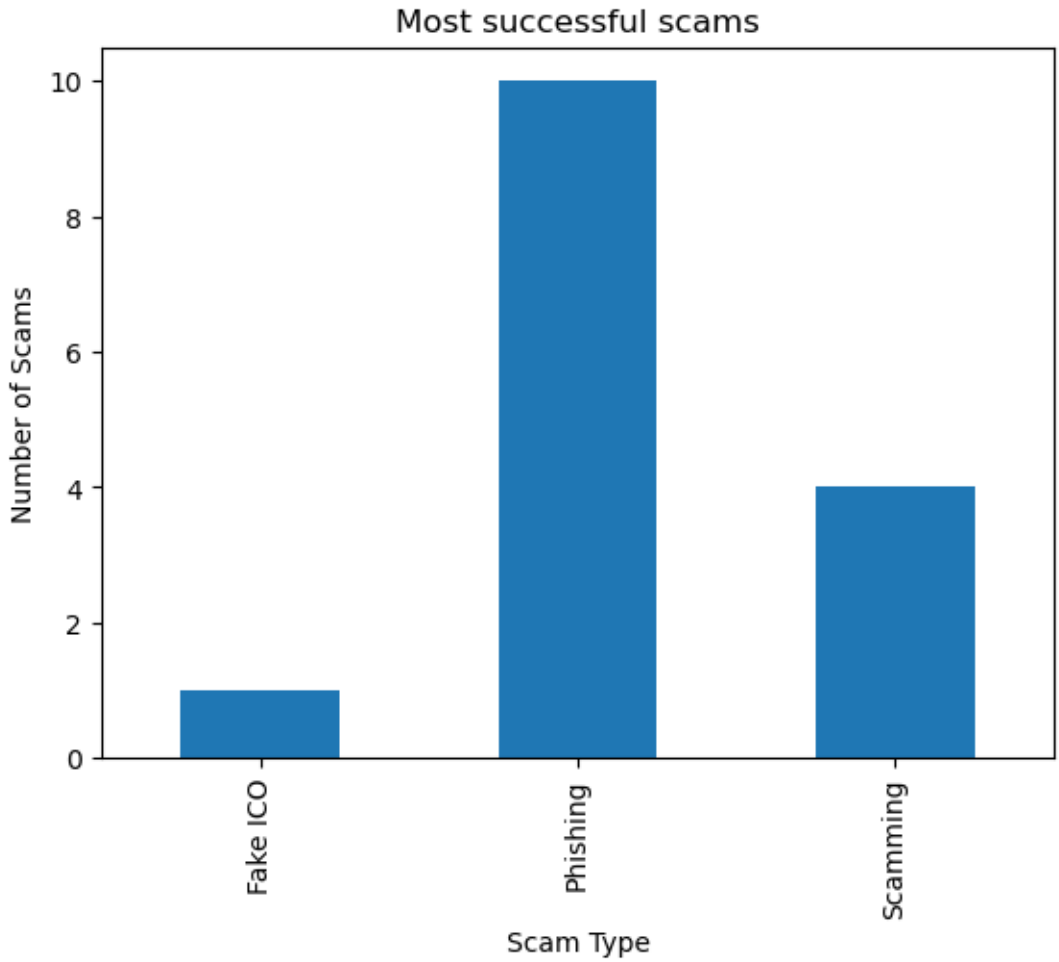
Scams Over time Graph:



Most Successful Scams:

	date	scam	val
0	5622	Scamming	1.67E+22
1	2135	Phishing	6.58E+21
2	90	Phishing	5.97E+21
3	2258	Phishing	3.46E+21
4	2137	Phishing	3.39E+21
5	2132	Scamming	2.43E+21
6	88	Phishing	2.07E+21
7	2358	Scamming	1.84E+21
8	2556	Phishing	1.80E+21
9	1200	Phishing	1.63E+21
10	2181	Phishing	1.16E+21
11	41	Fake ICO	1.15E+21
12	5820	Scamming	1.13E+21
13	86	Phishing	8.94E+20
14	2193	Phishing	8.83E+20

Most Successful Scams Graph:



Misc.

Here we aim to find how gas prices have changed over time for Ethereum transactions.

- The input file we use for this question is: **transactions.csv**
- Code: **gass_guzz.py**
- Command used to execute in terminal:

ccc create spark gasguzz.py -d -s

ccc method bucket cp bkt:eth_05_04_23_15:52:22/avg_gused.txt ~/BDNN_coursework/

ccc method bucket cp bkt:eth_05_04_23_16:22:22/avg_gprice.txt ~/BDNN_coursework/

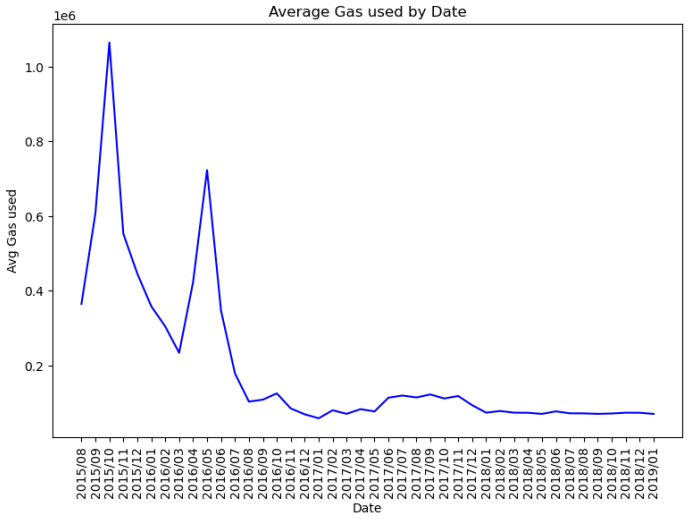
Methodology:

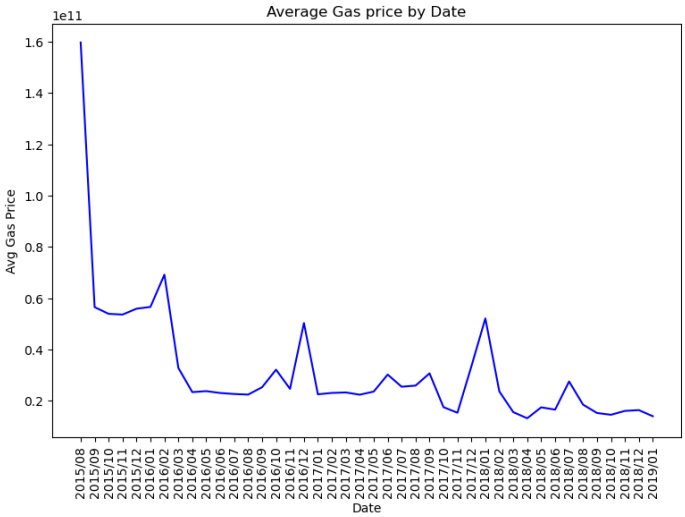
To obtain the average gas price change and the average gas used over time, the methodology involves processing two datasets, namely transactions.csv and contracts.csv. First, invalid transactions and contracts are filtered out based on format and missing values. Then, the average

gas price is calculated by mapping the date as the key and gas price and count as the value from the transactions dataset. The total gas price and count are obtained by reducing the mapped function using the reduceByKey function. The average gas price is obtained by dividing the total gas price by the total count and mapped to the date to get the average gas price change each month. The resulting output is stored in "avg_gas.txt" and plotted using Google Colab. Second, the transactions dataset is joined with the contracts dataset using the .join() function. The to_address is mapped as the key and the date and gas as the value from transactions.csv, and the address is mapped as the key and count as the value from contract.csv. The date is then mapped as the key and the gas used and count as the value, and the mapped function is reduced using reduceByKey to get the total gas used and count. Finally, the average gas used is obtained by dividing the total gas used by the total count and mapped to the date to get the average gas used each month. The resulting output is stored in "avg_gprice.txt" & “avg_gused.txt”.

Output file obtained: avg_gused.txt & avg_gprice.txt

Gas-guzzler data:





Findings:

From the graphs we observe that the average gas Price correlates with the average value of transactions by date as provided above.