ECS765 – BIG DATA PROCESSING COURSEWORK

Analysis of Ethereum Transactions and Smart Contracts

Omkar Bare (Student Number: 220459749)

School of Electronic Engineering and Computer Science, Queen Mary University of London
Mile End Road, London E1 4NS, UK

1. PART A – TIME ANALYSIS:

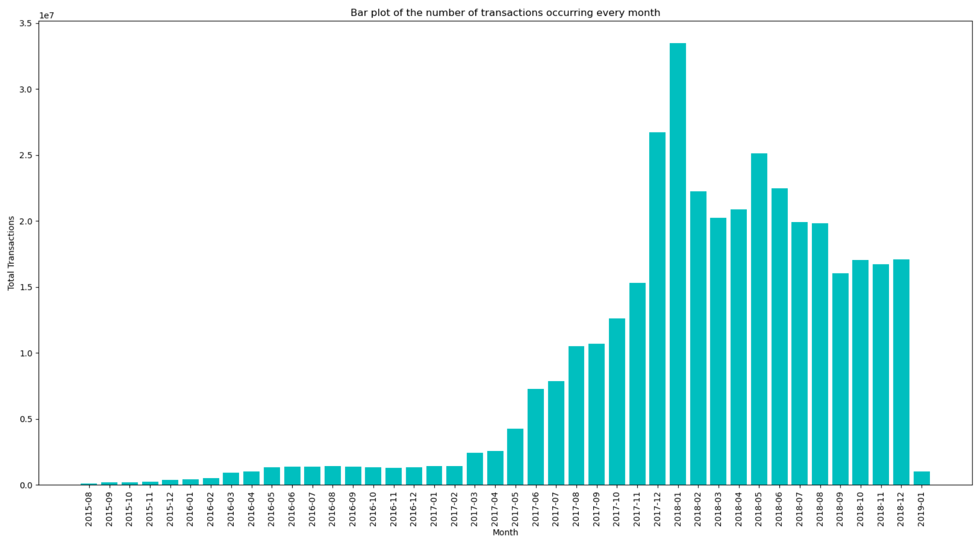
1.1. Bar plot showing the number of transactions occurring every month between the start and end of the dataset.

Code Methodology:

The 'check_transaction' function is used to filter out invalid transactions, and the resulting RDD is transformed using the 'map' function to create a tuple containing the month and year of each transaction and a count of 1. The 'reduceByKey' function is then used to aggregate the counts of transactions by month and year, resulting in an RDD containing pairs of keys (month/year) and their corresponding total transaction counts. Finally, the matplotlib library was used to plot the bar chart of number of transactions occurring every month.

Source Code Files attached:

1. parta.py
2. parta.ipynb
3. transactions_total.txt (output file used to create plot)
4. parta.png (bar plot output)



1.2. Bar plot showing the average value of transaction in each month between the start and end of the dataset.

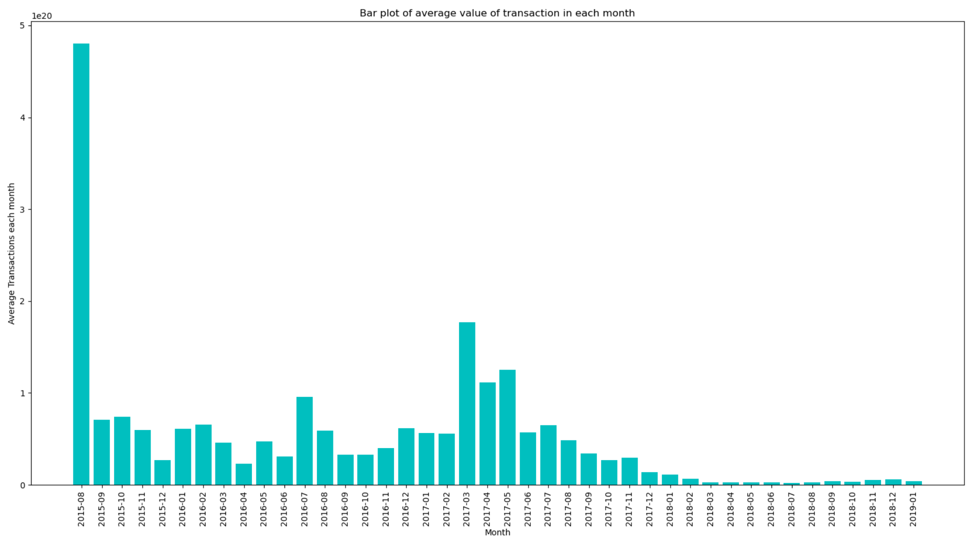
Code Methodology:

The 'check_transaction' function is used to filter out invalid transactions, and the 'mapping' function is used to extract the month and transaction value from each valid transaction and returns a key-value pair with the month as the key and a tuple of transaction value and count as the value. The 'reduceByKey' function is used to aggregate the transaction value and count tuples by month, summing the values and counts for each month.

Finally, the average transaction value is calculated by dividing the sum of transaction values by the count of transactions for each month and the matplotlib library was used to plot the bar chart of average value of transaction in each month.

Source Code Files attached:

1. parta2.py
2. parta2.ipynb
3. transactions_average.txt (output file used to create plot)
4. parta_2.png (bar plot output)



2. PART B - Top Ten Most Popular Services:

Code Methodology:

The code utilizes two filtering functions, 'check_transactions' and 'check_contracts,' to filter out invalid transaction records. These functions are applied to filter the transaction and contract records using the 'filter' function, resulting in new RDDs that only contain the valid records.

The valid transaction data is then mapped to extract the address and value information for each transaction. Similarly, the valid contract data is also mapped to extract only the address information. Next, the mapped transaction data is reduced by address, resulting in aggregated values for each address.

The reduced transaction data is then joined with the contract data based on the address to obtain the address and corresponding value of smart contracts. Finally, the address and value information of smart contracts is extracted, and the top 10 contracts by value are computed using the 'takeOrdered' function.

Source Code Files attached:

1. partb.py
2. top10_smart_contracts.txt

Top 10 Contracts by Value:

	Address	Value
0	0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444	84155363699941767867374641
1	0x7727e5113d1d161373623e5f49fd568b4f543a9e	45627128512915344587749920
2	0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef	42552989136413198919298969
3	0xbfc39b6f805a9e40e77291aff27aee3c96915bdd	21104195138093660050000000
4	0xe94b04a0fed112f3664e45adb2b8915693dd5ff3	15543077635263742254719409
5	0xabbb6bebfa05aa13e908eaa492bd7a8343760477	10719485945628946136524680
6	0x341e790174e3a4d35b65fdc067b6b5634a61caea	8379000751917755624057500
7	0x58ae42a38d6b33a1e31492b60465fa80da595755	2902709187105736532863818
8	0xc7c7f6660102e9a1fee1390df5c76ea5a5572ed3	1238086114520042000000000
9	0xe28e72fcf78647adce1f1252f240bbfaebd63bcc	1172426432515823142714582

3. PART C - Top Ten Most Active Miners:

Code Methodology:

The 'clean_lines' RDD is generated by filtering out the invalid blocks from the 'blocks.csv' data using the 'check_blocks' function. The 'blocks_mapped' RDD is then created by applying the 'features_blocks' function to the 'clean_lines' RDD to extract the required features, which are the miner's address and block size.

Then the 'blocks_reduced' RDD is obtained by reducing the 'blocks_mapped' RDD by the key (miner's address) using the 'reduceByKey' function and summing up the block sizes. Finally, the 'takeOrdered' function is used to obtain the top 10 miners by selecting the 10 miners with the largest block sizes from the 'blocks_reduced' RDD in descending order.

Source Code Files attached:

1. partc.py
2. top10_miners.txt

Top 10 Most Active Miners:

	Miner	Block Size
0	0xea674fdde714fd979de3edf0f56aa9716b898ec8	17453393724
1	0x829bd824b016326a401d083b33d092293333a830	12310472526
2	0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c	8825710065
3	0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5	8451574409
4	0xb2930b35844a230f00e51431acae96fe543a0347	6614130661
5	0x2a65aca4d5fc5b5c859090a6c34d164135398226	3173096011
6	0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb	1152847020
7	0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01	1134151226
8	0x1e9939daaad6924ad004c2560e90804164900341	1080436358
9	0x61c808d82a3ac53231750dadc13c777b59310bd9	692942577

4. PART D – Scam Analysis:

4.1. Popular Scams:

Code Methodology:

First the scams.json file in converted to scams.csv file using ‘convert_json_to_csv.py’ file.

Then in the ‘t4scams.py’ the 'clean_transactions' dataset is created by filtering out invalid transaction records using the 'is_valid_transaction' function. This dataset is then mapped to key-value pairs where the key is the address and the value is the transaction amount. Similarly, the 'clean_scams' dataset is created by filtering out invalid scam records using the 'is_valid_scam' function. This dataset is then mapped to key-value pairs where the key is a tuple of the address and the scam type, and the value is a tuple of the scam ID and the scam timestamp. The 'clean_scams' dataset is then flattened using 'flatMap' function to create multiple records with the same scam ID but different Ethereum addresses. The ‘join’ function is then used to join the two datasets on the Ethereum address, and ‘reduceByKey’ function is used to aggregate the transaction amounts by the scam ID.

Then, a new 'clean_scams_new' dataset is created by filtering out invalid scam records using the 'is_valid_scam' function. This dataset is then mapped to key-value pairs where the key is the address and the value is the scam type. Similar to above, a new 'clean_transactions_new' dataset is created by filtering out invalid transaction records using the 'is_valid_transaction' function. This dataset is then mapped to key-value pairs where the key is the address and the value is a tuple of the transaction timestamp and amount. The two datasets are then joined on the address, and the resulting dataset is mapped to key-value pairs where the key is a tuple of the transaction timestamp and scam type, and the value is the transaction amount. Finally, ‘reduceByKey’ function is used to aggregate the transaction amounts by the timestamp and scam type.

Source Code Files attached:

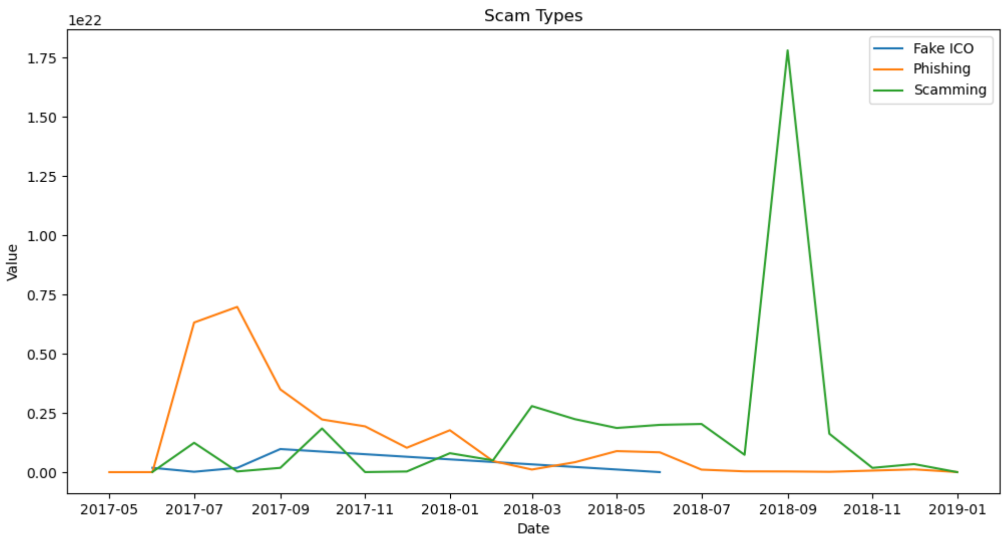
1. t4scams.py
2. convert_json_to_csv.py
3. top_scams.txt (output file)
4. ether_time.txt (output file)
5. scams.csv (input file)
6. scams.json (input file)

Results:

most lucrative scam:

ID: ‘5622’, Scan Type: ‘Scamming’, Value: 1.6709083588072844e+22

	ID	Value	Scam Type
0	5622	1.670908e+22	Scamming
1	2135	6.583972e+21	Phishing
2	90	5.972590e+21	Phishing
3	2258	3.462808e+21	Phishing
4	2137	3.389914e+21	Phishing
5	2132	2.428075e+21	Scamming
6	88	2.067751e+21	Phishing
7	2358	1.835177e+21	Scamming
8	2556	1.803047e+21	Phishing
9	1200	1.630577e+21	Phishing



In 2017, phishing and scamming were the primary types of scams, with phishing comprising the majority of them, while fake ICOs occurred less frequently. Although the number of phishing scams decreased throughout the year, scamming scams remained consistent. In 2018, there was a noticeable increase in scamming scams compared to phishing scams, along with fake ICOs occurring less frequently. The trend suggests a shift towards scamming scams from phishing scams over the years.

4.2 Wash Trading:

Wash trading is a fraudulent practice aimed at creating fake trading volume and manipulating prices. One method of achieving this is by self-trading, where an individual trades with themselves using two different accounts. This creates a misleading impression of increased trading activity on the exchange. Self-trading can be used by market manipulators to artificially inflate prices or generate exchange fees. Additionally, it may create an illusion of liquidity that could attract more traders to the platform.

Code Methodology:

The 'check_transactions()' function is used to filter out invalid trasactions. Then, using the map() operation, only the required columns of the CSV file are extracted and a new RDD is created. This RDD is then converted into a DataFrame and filtered to only include transactions where the "from_address" column is equal to the "to_address" column. The resulting RDD is then mapped to create tuples containing the addresses and value, which are then reduced by key using the 'reduceByKey()' operation. Finally, the top 10 pairs of addresses with the highest sum of float values are obtained.

Source Code Files attached:

1. washtrading.py
2. top10_washtrade.txt

Results:

Trader with highest value of self-trade:

address: 0x02459d2ea9a008342d8685dae79d213f14a87d43

Value: 1.9548531332493194e+25

Top 10 Self Trades:

	Address	Values
0	0x02459d2ea9a008342d8685dae79d213f14a87d43	1.954853e+25
1	0x32362fbfff69b9d31f3aae04faa56f0edee94b1d	5.295491e+24
2	0x0c5437b0b6906321cca17af681d59baf60afe7d6	2.377153e+24
3	0xdb6fd484cfa46eeeb73c71edee823e4812f9e2e1	4.154974e+23
4	0xd24400ae8bfebb18ca49be86258a3c749cf46853	2.270001e+23
5	0x5b76fbe76325b970dbfac763d5224ef999af9e86	7.873327e+22
6	0xdd3e4522bdd3ec68bc5ff272bf2c64b9957d9563	5.790176e+22
7	0x005864ea59b094db9ed88c05ffba3d3a3410592b	3.719900e+22
8	0x4739928c37159f55689981b10524a62397a65d77	3.023900e+22
9	0xb8326d2827b4cf33247c4512b72382f4c1190710	2.457200e+22