ECS765P - Big Data Processing

Coursework Report

Analysis of Ethereum Transactions and Smart Contracts

Name: Gitesh Deshmukh

Student ID: 220900436

PART A - Time Analysis

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

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

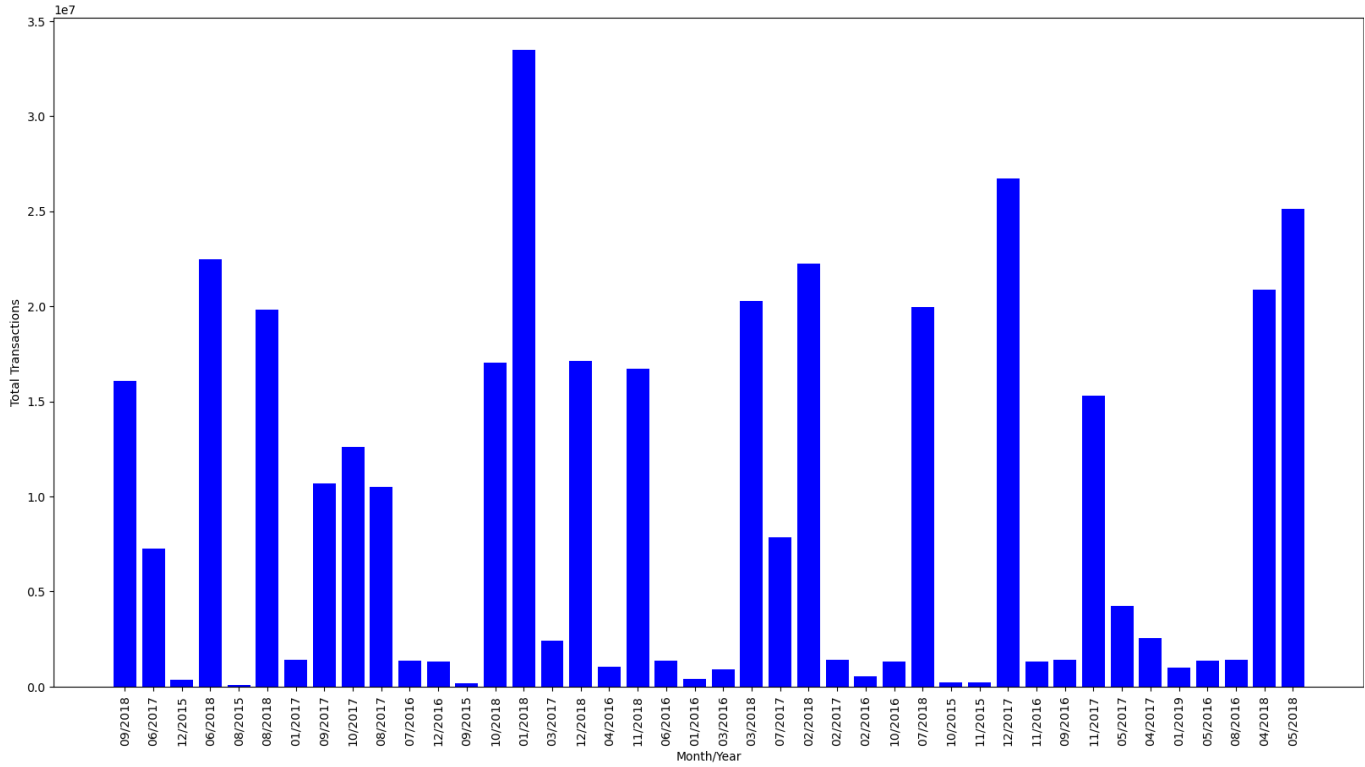
Note: As the dataset spans multiple years and you are aggregating together all transactions in the same month, make sure to include the year in your analysis.

Note: Once the raw results have been processed within Spark you may create your bar plot in any software of your choice (excel, python, R, etc.)

Explanation:

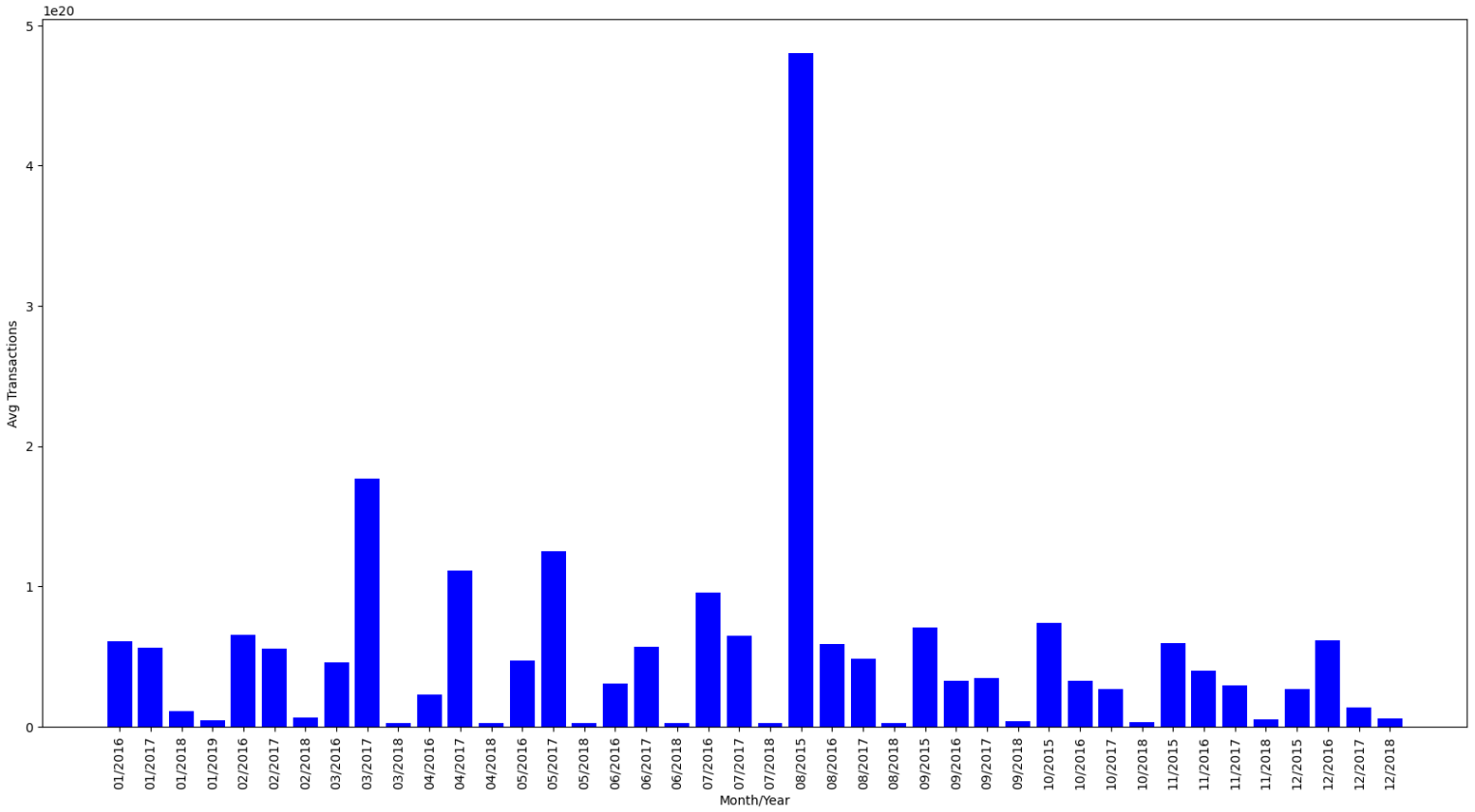
The transactions.csv is the input data file in this Time Analysis process. For the first task to show the number of transactions occurring every month between the start and end of the dataset, I extracted the Date and the Total number of transactions and defined the transactions_time function where I ran the if statement to filter out lines that do not contain 15 fields and converts 12th field to an integer to ensure lines containing the valid transaction. Then to count the total number of transactions for each month/year the reduceByKey function is used. The output is stored in total_time_transaction.txt. Following this, the output data is plotted using matplotlib in partavg.ipynb file where plot is generated.

The output bar graph is shown below which is generated using matplotlib file.



Bar Plot : Month/Year vs Total Transactions

For the second task in Part A: Time Analysis, I got the average value of transactions and the time. Again, by using reduceByKey whic averages two values and returns as one as it helps to concise the values, I got the total values and number of transactions. Then, aligned the data with the proper date format and displayed average values. The py file is named as partavg.py. The output is saved in avg_transaction.txt while the bar graph is plotted in Avg Transaction vs Month/Year.



Bar Plot : Month/Year vs Avg Transactions

PART B - Top Ten Most Popular Services

Evaluate the top 10 smart contracts by total Ether received. You will need to join **address** field in the contracts dataset to the **to_address** in the transactions dataset to determine how much ether a contract has received.

Explanation:

In this code, the input data is taken from transactions.csv and Contracts.csv which is already provided. Using the Transaction and Contracts dataset, the address and the value is mapped respectively as per field location. Further, reduceByKey function is used and the values stored in tr_reducing. Then, address and value are mapped together and stored in address_value. The top 10 smart contract values is generated using takeOrdered() function which returns all the top 10 addresses with highest values. The generated output after running in the terminal is stored in top_ten_smart_contracts.txt.

RANK	ADDRESS	ETHEREUM VALUE
1.	"0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444"	84155363699941767867374641
2.	"0x7727e5113d1d161373623e5f49fd568b4f543a9e"	45627128512915344587749920
3.	"0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef"	42552989136413198919298969
4.	"0xbfc39b6f805a9e40e77291aff27aee3c96915bdd"	21104195138093660050000000
5.	"0xe94b04a0fed112f3664e45adb2b8915693dd5ff3"	15543077635263742254719409
6.	"0xabbb6bebfa05aa13e908eaa492bd7a8343760477"	10719485945628946136524680
7.	"0x341e790174e3a4d35b65fdc067b6b5634a61caea"	8379000751917755624057500
8.	"0x58ae42a38d6b33a1e31492b60465fa80da595755"	2902709187105736532863818
9.	"0xc7c7f6660102e9a1fee1390df5c76ea5a5572ed3"	1238086114520042000000000
10.	"0xe28e72fcf78647adce1f1252f240bbfaebd63bcc"	1172426432515823142714582

PART C - Top Ten Most Active Miners

Evaluate the top 10 miners by the size of the blocks mined. This is simpler as it does not require a join. You will first have to aggregate **blocks** to see how much each miner has been involved in. You will want to aggregate **size** for addresses in the **miner** field. This will be similar to the wordcount that we saw in Lab 1 and Lab 2. You can add each value from the reducer to a list and then sort the list to obtain the most active miners.

Explanation:

For this Part C, only blocks dataset is used which is stored in csv format. The miner data is mapped by using clean_contract_lines where filter data is been stored. Again, reduceByKey is used and stored and the output is saved in “Top_Miners.txt”

Output:

RANK	MINER	BLOCK SIZE
1.	"0xea674fdde714fd979de3edf0f56aa9716b898ec8"	17453393724
2.	"0x829bd824b016326a401d083b33d092293333a830"	12310472526
3.	"0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c"	8825710065
4.	"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5"	8451574409
5.	"0xb2930b35844a230f00e51431acae96fe543a0347"	6614130661
6.	"0x2a65aca4d5fc5b5c859090a6c34d164135398226"	3173096011
7.	"0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb"	1152847020
8.	"0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01"	1134151226
9.	"0x1e9939daaad6924ad004c2560e90804164900341"	1080436358
10.	"0x61c808d82a3ac53231750dadc13c777b59310bd9"	692942577

PART D - Data exploration

Scam Analysis

(i) Popular Scams: Utilising the provided scam dataset, what is the most lucrative form of scam? How does this change throughout time, and does this correlate with certain known scams going offline/inactive? To obtain the marks for this category you should provide the id of the most lucrative scam and a graph showing how the ether received has changed over time for the dataset.

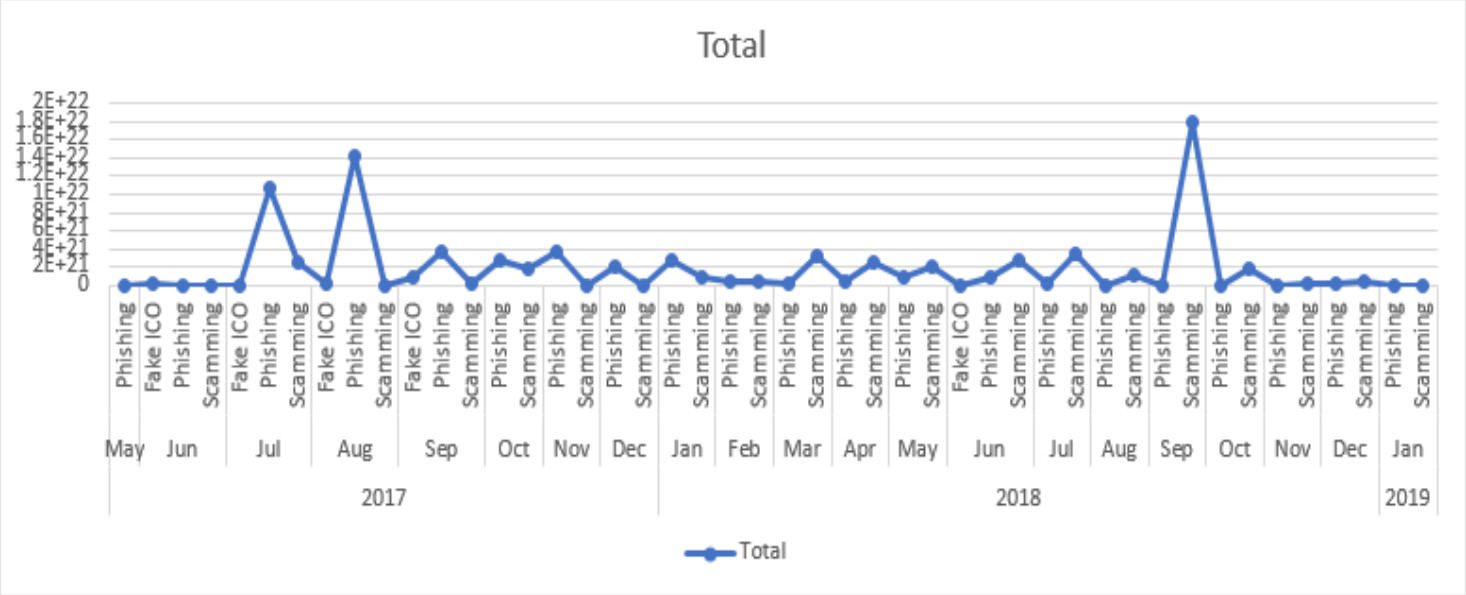
Explanation:

The input data we have used here is transactions.csv and scams.json. I checked the valid transaction and valid scam records using two check functions using if statement. The filter data values are stored in two variables respectively. For lucrative scam data purposes, I used the join function which stores the values in a variable(joins1) after mapping the address and category. The reduceByKey function enables to get the total value categorically. The top 15 lucrative scams is generated using takeOrdered() function and the output text file after running in terminal is saved in “the_lucrative_scam.txt” file.

Output:

Rank	Scam ID	Scam Type	Total Ether Profited
1.	5622	Scamming	1.6709083588072808e+22
2.	2135	Phishing	6.583972305381559e+21
3.	90	Phishing	5.972589629102411e+21
4.	2258	Phishing	3.462807524703738e+21
5.	2137	Phishing	3.389914242537183e+21
6.	2132	Scamming	2.428074787748575e+21
7.	88	Phishing	2.0677508920135265e+21
8.	2358	Scamming	1.8351766714814893e+21
9.	2556	Phishing	1.803046574264181e+21
10.	1200	Phishing	1.63057741913309e+21
11.	2181	Phishing	1.1639041282770013e+21
12.	41	Fake ICO	1.1513030257909173e+21
13.	5820	Scamming	1.1339734671862086e+21
14.	86	Phishing	8.944561496957756e+20
15.	2193	Phishing	8.827100174717214e+20

Total Ether received over time:



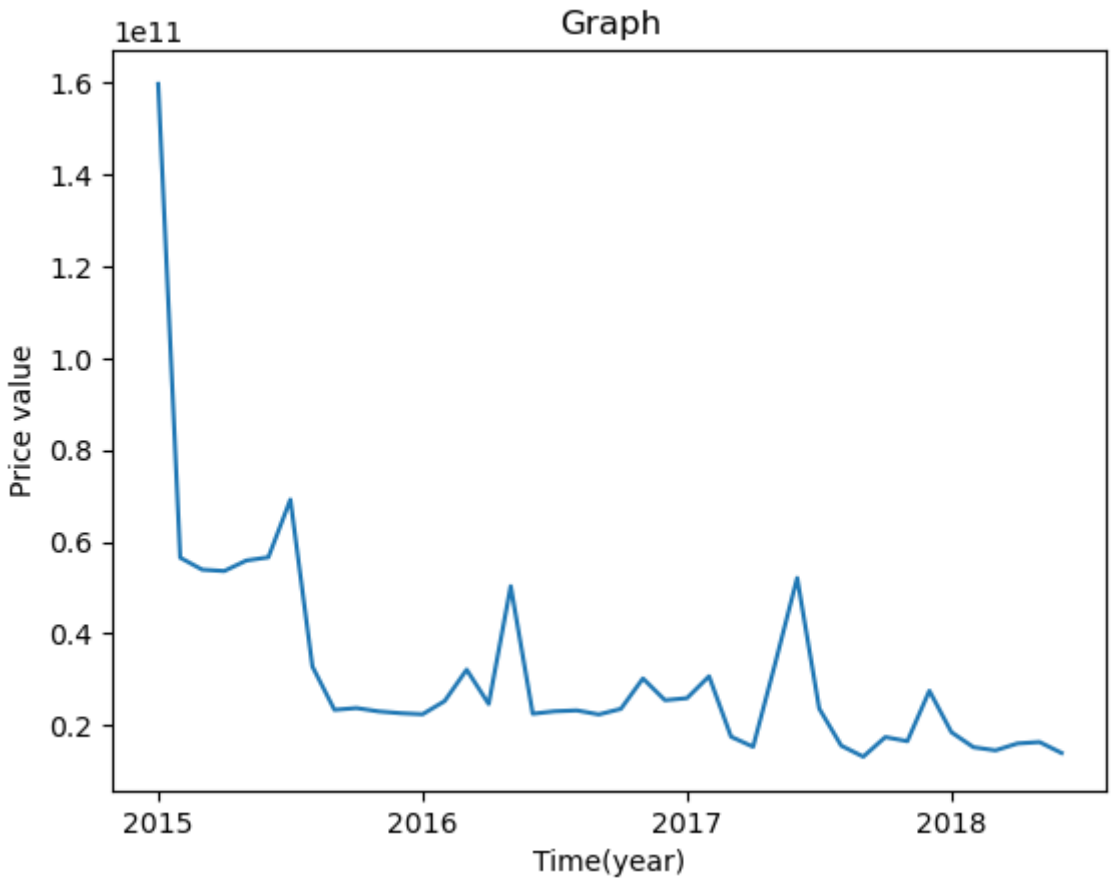
Miscellaneous Analysis

(ii) Gas Guzzlers: For any transaction on Ethereum a user must supply gas. How has gas price changed over time? Have contracts become more complicated, requiring more gas, or less so? How does this correlate with your results seen within Part B. To obtain these marks you should provide a graph showing how gas price has changed over time, a graph showing how gas used for contract transactions has changed over time and identify if the most popular contracts use more or less than the average gas_used.

Explanation:

The datasets used for this miscellaneous analysis are from transactions.csv and contracts.csv. In the Gus Guzzler analysis, two things are achieved one being average gas price change over time and other average gas used over the time span. The mapping function maps the block timestamp data which is transaction data to the date format and returns the tuple for gas price and count from the transactions.csv dataset. To get total average gas price, the date is mapped as a key and gp as gas price. The reduceByKey reduces the mapped values by key and stores it in t_reducing variable and sorted using sortByKey function. The output file is saved as avg_gasprice.txt .

Output:



Similarly, The gas used over the time is calculated by mapping to_address as a key, while date and the float value of g as value which is returned in mapping2 function. Since both datasets are used, I have joined both datasets using join function. The reduceByKey performs the similar for gas used and stores it in the t_reducing1. The output file is saved as avg_gasused.txt.

Output: