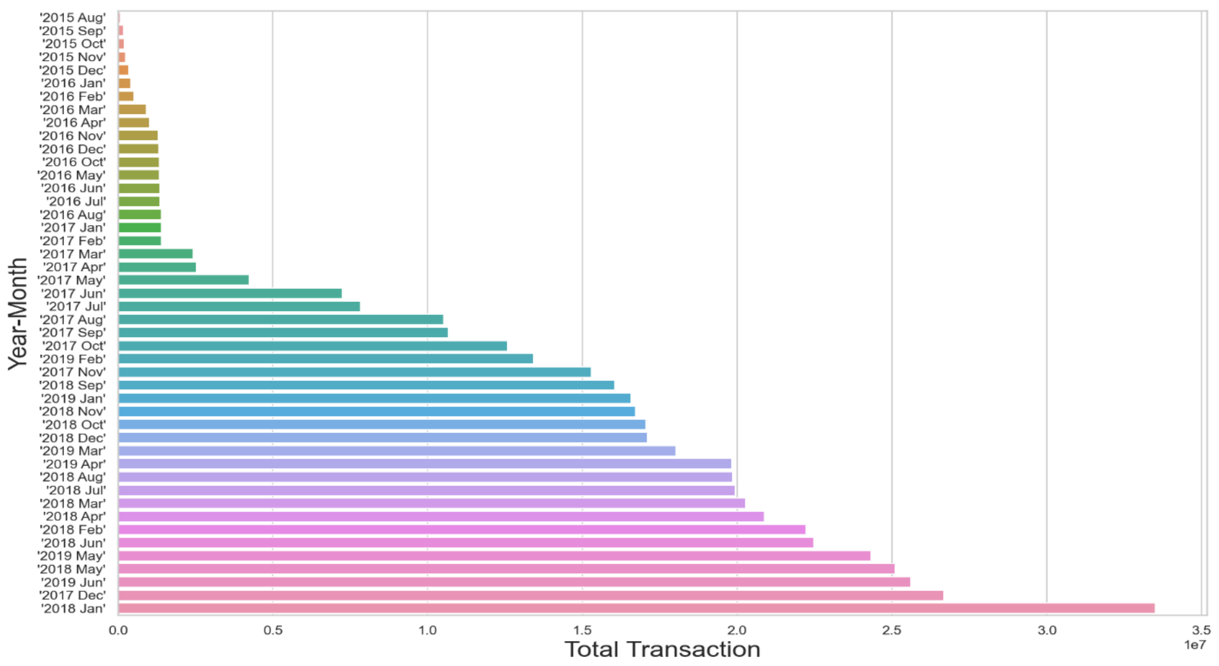
ECS640U-ECS765P- BIG DATA PROCESSING- 2020-2021 Coursework

I have written all of the scripts in pySpark. In addition, their outputs are available in the outputs folder.

Part A. TIME ANALYSIS:

In the "numberoftransactions.py" script, the total number of transactions are aggregated by each month in the dataset and the result is shown below:

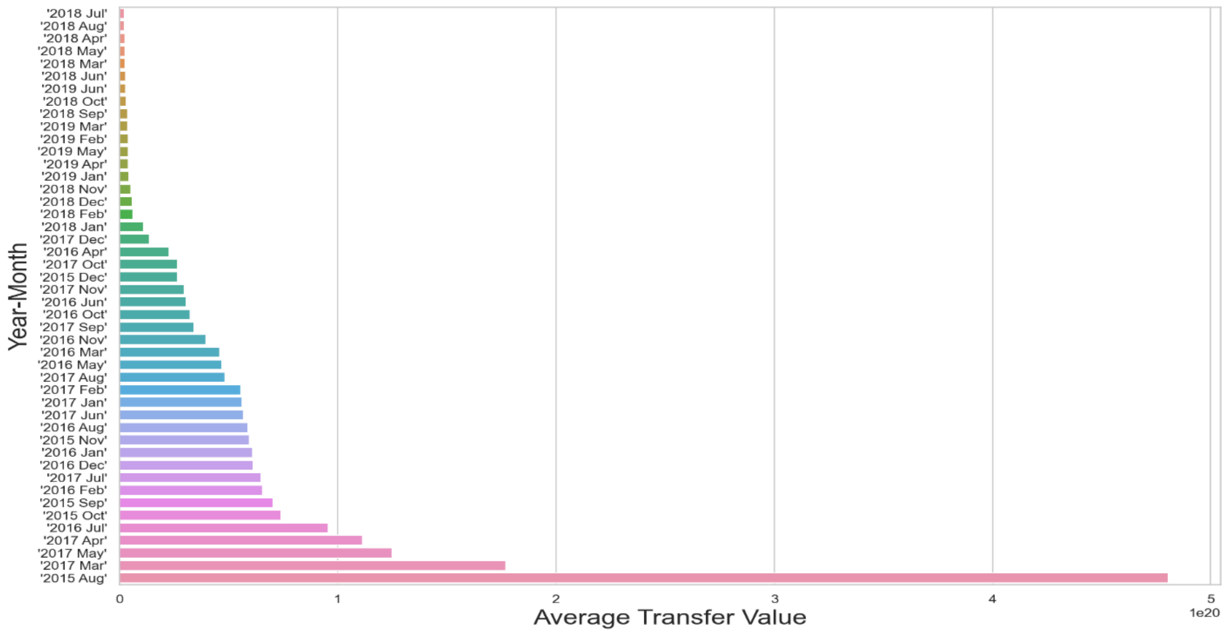


I commenced by loading the transactions dataset and filtering out the bad lines. Then I mapped every line to use the 6th field as key (represents the date year-month) with a value of (1) to represent the counter. Then I counted the occurrence of every key using the reduceByKey function. Then used the output to generate the given plot after it was sorted using Seaborn library on my local machine.

Corresponding id for the job:

http://andromeda.eecs.qmul.ac.uk:8088/cluster/app/application_1607539937312_8833

In the “Averagetransfervalue.py” script, the average value of transaction is calculated in each month between the start and end of the dataset.



I commenced by loading the dataset and filtering out the bad lines. Then I mapped every line to use the 6th field as key (represents the date year-month) and, as values, the 3rd field to represent the transaction-value and (1) to represent the counter. Then I summed the values of every key using the reduceByKey function. Then I divided the summed values by the summed counter for every key using the mapValues function.

Corresponding id for the job:

http://andromeda.eecs.qmul.ac.uk:8088/cluster/app/application_1607539937312_8895

Part B. TOP TEN MOST POPULAR SERVICES:

In the “Averagetransfervalue.py” script, the top 10 services with the highest amounts of Ethereum received for smart contracts are yielded.

0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444	8.415510081e+25
0xfa52274dd61e1643d2205169732f29114bc240b3	4.57874844832e+25
0x7727e5113d1d161373623e5f49fd568b4f543a9e	4.56206240014e+25
0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef	4.31703560923e+25
0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8	2.7068921582e+25
0xbfc39b6f805a9e40e77291aff27aee3c96915bdd	2.11041951381e+25
0xe94b04a0fed112f3664e45adb2b8915693dd5ff3	1.55623989568e+25
0xbb9bc244d798123fde783fcc1c72d3bb8c189413	1.19836087292e+25
0xabbb6bebfa05aa13e908eaa492bd7a8343760477	1.17064571779e+25
0x341e790174e3a4d35b65fdc067b6b5634a61caea	8.37900075192e+24

I commenced by loading the transactions and contracts datasets and filtering out the bad lines from both. Then from the transactions dataset I mapped every line to use the 2nd field as key (represents the transaction to_address) and, as values, the 3rd field that represents the transaction-value. Then I summed the values of every key using the reduceByKey function.

Also, from the contract dataset I mapped every line to use the (0) field as key (represents the contracts’ address) and, as values, the value of (1) to represent the counter.

Then I performed a lefouterjoin on the two datasets using the keys as point of joining because they represent addresses. Then I filtered out rows that have “None” values from the joined tables. Then I mapped the joined table to just two fields (addresses, total_Ether_received) then I took the top 10 rows as output.

Corresponding id for the job:
http://andromeda.eecs.qmul.ac.uk:8088/cluster/app/application_1607539937312_8954

Part C. TOP TEN MOST ACTIVE MINERS:

The “toptenmostactiveminers.py” script, Evaluates the top 10 miners by the size of the blocks mined.

0xea674fdde714fd979de3edf0f56aa9716b898ec8	23989401188.0
0x829bd824b016326a401d083b33d092293333a830	15010222714.0
0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c	13978859941.0
0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5	10998145387.0
0xb2930b35844a230f00e51431acae96fe543a0347	7842595276.0
0x2a65aca4d5fc5b5c859090a6c34d164135398226	3628875680.0
0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01	1221833144.0
0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb	1152472379.0
0x1e9939daaad6924ad004c2560e90804164900341	1080301927.0
0x61c808d82a3ac53231750dadc13c777b59310bd9	692942577.0

I commenced by loading the Blocks dataset and filtering out the bad lines. Then I mapped every line to use the 2nd field as key (represents the miners address) and, as values, the 4th field to represent the block size the miner has mined. Then I summed the values of every key using the reduceByKey function. then I took the top 10 rows as output.

Corresponding id for the job:

http://andromeda.eecs.qmul.ac.uk:8088/cluster/app/application_1607539937312_8978

Part D:

SCAM ANALYSIS:

Most lucrative forms of scams:

“scamanalysis.py”

'Scamming'	3.8363145949412604e+22
'Phishing'	2.695757418124426e+22
'Fake ICO'	1.35645756688963e+21

After parsing the Json file to csv format I commenced by loading the transactions and scams datasets and filtering out the bad lines from the transactions dataset. Then from the transactions dataset I mapped every line to use the 2nd field as key (represents the transaction to_address) and, as values, the 3rd field that represents the transaction-value. Then I summed the values of every key using the reduceByKey function.

Then from the scams dataset I mapped every line to use the (0) field as key (represents the transaction scam type) and, as values, every field from the 2nd field because they represents the a addresses of the scams. Them I used flatMapValues so that every address corresponds to its scam type. Then I used the distinct function to remove duplicates as I noticed the scams dataset contained duplicates data. Then I rearranged the columns in the rows so that the scam address is the key.

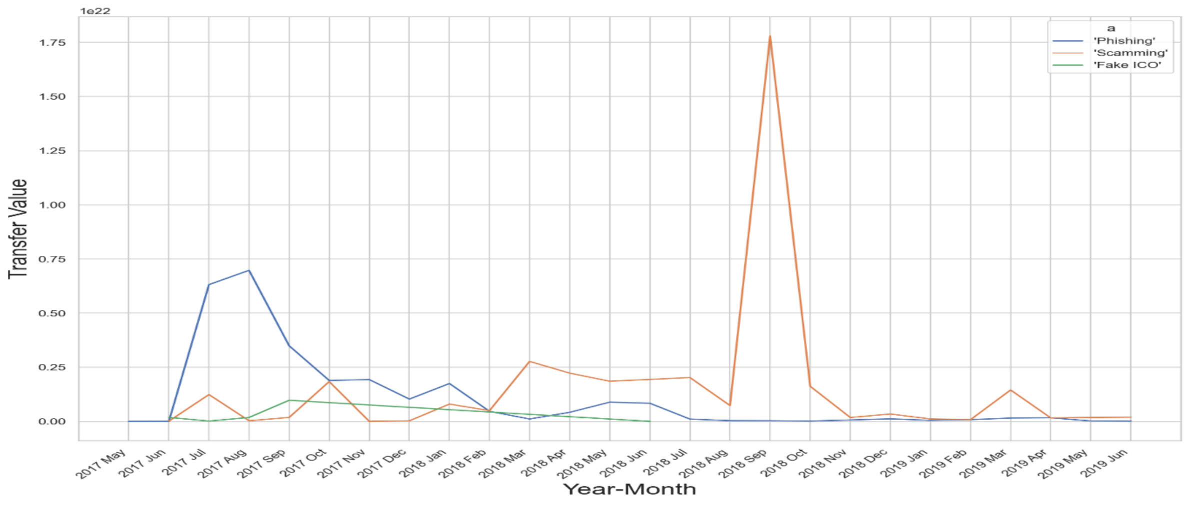
Then I performed a lefouterjoin on the two datasets using the keys as point of joining because they represent addresses. Then I filtered out rows that have “None” values from the joined tables. Then I rearranged the columns in the rows so that the scam type is the key and the value is the scam transaction-value. Then I summed the values of every key using the reduceByKey function to get the total for every scam type. Then I sorted the output.

Corresponding id for the job:

http://andromeda.eecs.qmul.ac.uk:8088/cluster/app/application_1607539937312_8987

Scams through time:

"scamanalysisthroughtime.py"



In July of 2017, phishing attacks were the most profitable type of scam. In September of 2017, fake ICO was the most profitable type of scam. In September of 2018, scamming attacks were the most profitable type of scam.

According to this websites https://news.bitcoin.com/data-shows-ethereum-is-the-cryptocurrency-of-choice-for-scams/ that after 2018 scamming on Ethereum has been in a decline in addition, from late 2016 through the end of 2018, Chainalysis has identified over 2,000 scam addresses on Ethereum. This might indicate that these addresses have been rendered inactive/offline at the end of 2018, which correlates with above plot.

This script roughly the same as the previous one.

I commenced by loading the transactions and scams datasets and filtering out the bad lines from the transactions dataset. Then from the transactions dataset I mapped every line to use the 2nd field as key (represents the transaction to_address) and, as values, the 3rd field that represents the transaction-value and the the 6th field as key (represents the date year-month). Then I summed the values of every key using the reduceByKey function.

Then from the scams dataset I mapped every line to use the (0) field as key (represents the transaction scam type) and, as values, every field from the 2nd field because they represents the a addresses of the scams. Them I used flatMapValues so that every address corresponds to its scam type. Then I used the distinct function to remove duplicates as I noticed the scams dataset contained duplicates data. Then I rearranged the columns in the rows so that the scam address is the key.

Then I performed a lefouterjoin on the two datasets using the keys as point of joining because they represent addresses. Then I filtered out rows that have “None” values from the joined tables. Then I rearranged the columns in the rows so that the scam type and the date is the key and the value is the scam transaction-value. Then I summed the values of every key using the reduceByKey function to get the total for every scam type. Then I sorted the output.

Corresponding id for the job:
http://andromeda.eecs.qmul.ac.uk:8088/cluster/app/application_1607539937312_9014

MACHINE LEARNING:

“machinelearning.py” The forecast output of this script is available in the output folder.

To forecast the prices I split up the prices dataset to two part:

1st part is from the beginning to 30/06/2019.

2nd part is from the 01/07/2019 to the end.

I used the 1st to train my model and the 2nd to predict.

I used these columns as features "Open (USD)”, “High (USD)", "Low (USD)"

And this as the target column "Closing Price (USD)"

I started by loading both datasets and making them both datasets into dataframes. Then I trained on the 1st dataset. Then I used the 2nd dataset to predict. In addition, I added a column to the prediction output dataframe which I called “**MAPE Accuracy**” which contains the MAPE accuracy between the actual price and the predicted price in every day since “01/07/2019” the statistical summary of the “**MAPE Accuracy**” column is recorded in the table below.

summary	**MAPE Accuracy**
count	488
mean	98.75692080870282
stddev	1.5496267831338013
min	76.55970176250722
max	99.9948548910462

Also in the script I also perfomed a “sum” on the “prediction” column and the “Closing Price (USD)” then I calculated the MAPE accuracy between the two sums which yielded the result below:

MAPE total accuracy : 99.5986757169%

Average accuracy between “01/11/2020” and end of June 2019 is 98.7% meaning high accuracy in this period and according to the output the model remained with good accuracy to the last date.

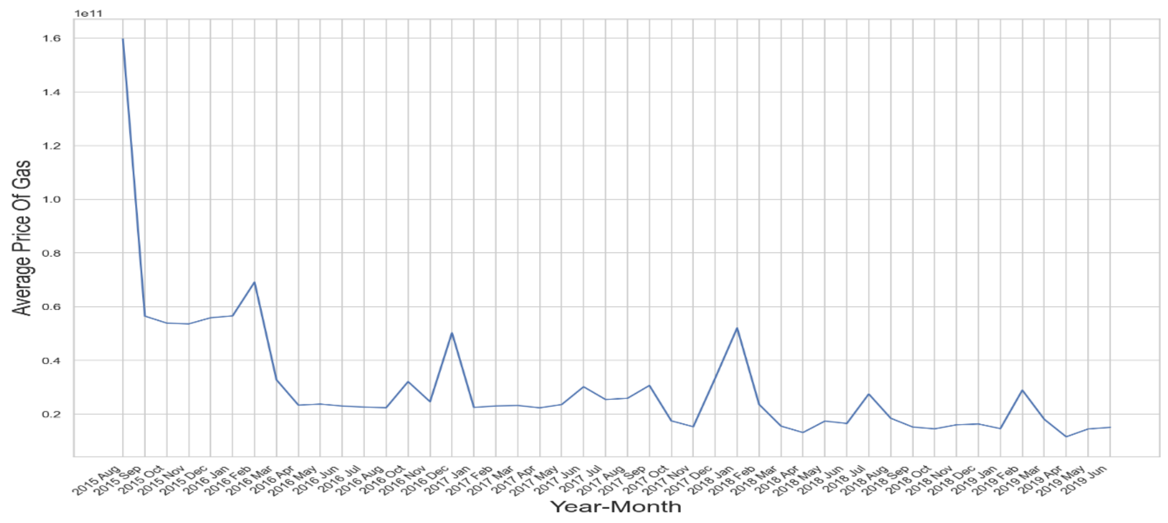
Corresponding id for the job:
http://andromeda.eecs.qmul.ac.uk:8088/cluster/app/application_1607539937312_9133

MISCELLANEOUS ANALYSIS:

Gas Guzzlers:

Transactions gas price vs time

In “gasguzzlers.py”, the average price of gas is calculated for each month by: total sum of gas prices / total number of transactions.

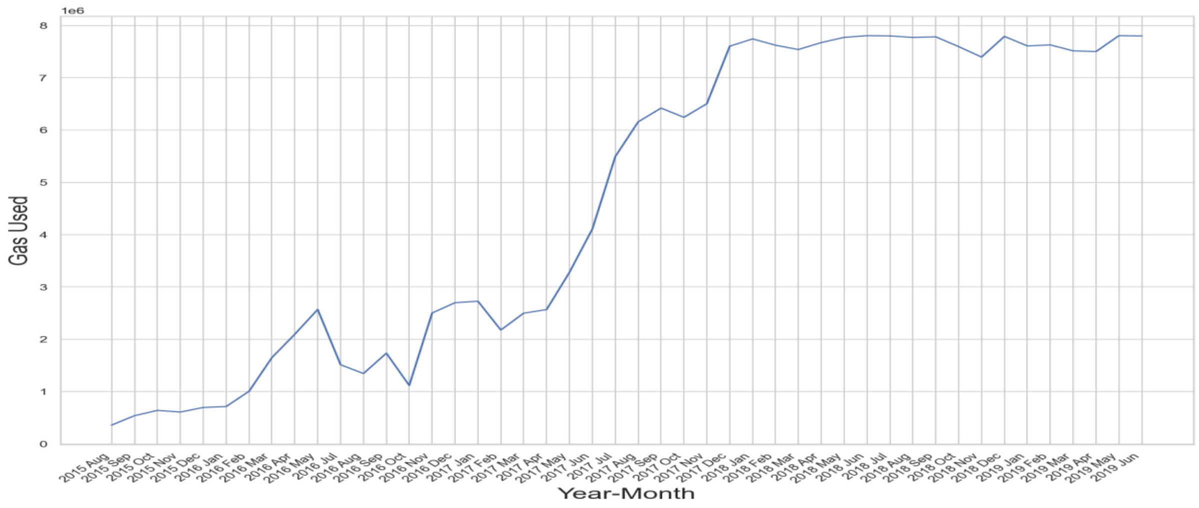


The average price of gas has gradually decreased with peaks during the first few months of each year.

This plot is obtained by loading the transaction dataset and filtering out the bad lines. Then I mapped every line to use the 6th field as key (represents the date year-month) and, as values the values of the 5th field to represent the gas_price and (1) to represent the counter. Then I summed the values of every key using the reduceByKey function. Then I divided the summed values by the summed counter for every key using the mapValues faunction.

Contract gas vs time

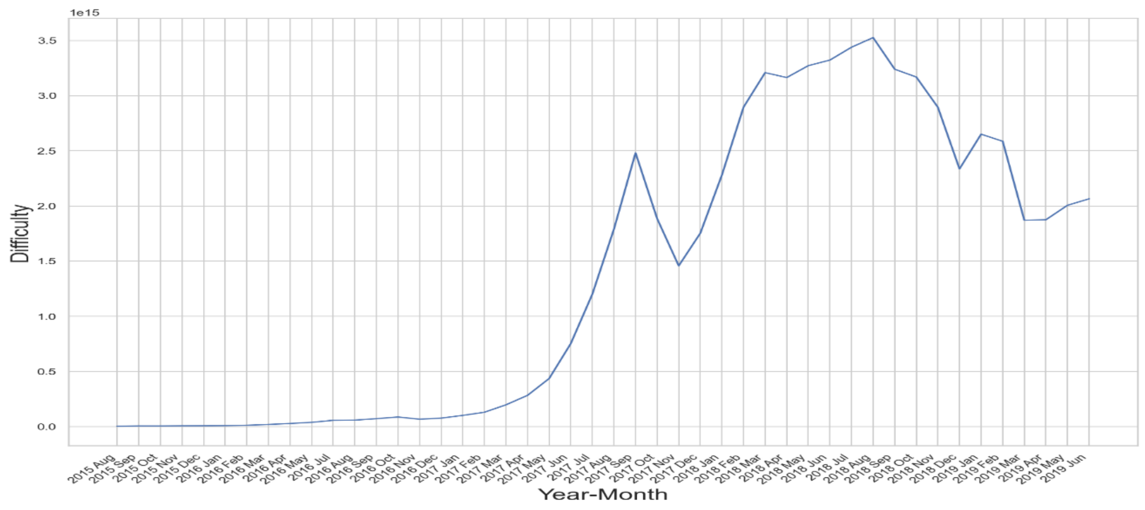
In “gasguzzlers.py”, the same approach for transactions is taken for smart contracts as well, with one alteration: in order to make sure a block represents a smart contract, it needs to be joined with the contracts dataset. Therefore, I joined the blocks dataset and the contracts dataset. Because the block id is unique and present in both datasets, it is used as the joining key. In addition, I mapped the joined the tables so that the date is the Key and the difficulty and the gas_used are values. The I produced the two following plots from the outputs.



Contracts have been requiring more gas since the start of Ethereum, and it appears that the gas used has reached a plateau.

Contract complexity vs time

The complexity of a transaction is shown in its difficulty level, which can be extracted from the blocks dataset.



Contracts have become more complicated with peaks in mid-late 2017 and mid-2018, followed by a gradual decrease.

The required gas for a contract apears to be strongly correlated with the contract's difficulty:

df['b'].corr(df['c']) = 0.9376

with ‘b’ refers to Difficulty, and ‘c’ refers to Difficulty gas_used.

With higher complexity, more gas would be required, so the most popular services might have been complex contracts that need to have a lot of gas to be mined. Therefore, they receive a lot of ether to be fuelled.

Corresponding id for the job:
http://andromeda.eecs.qmul.ac.uk:8088/cluster/app/application_1607539937312_9141