

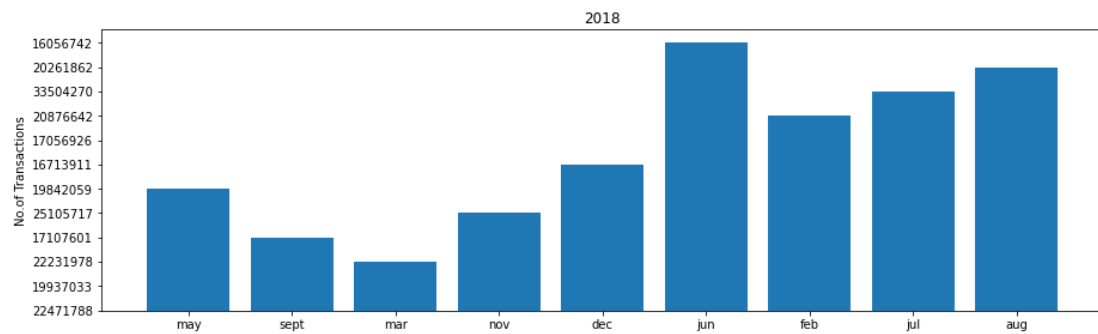
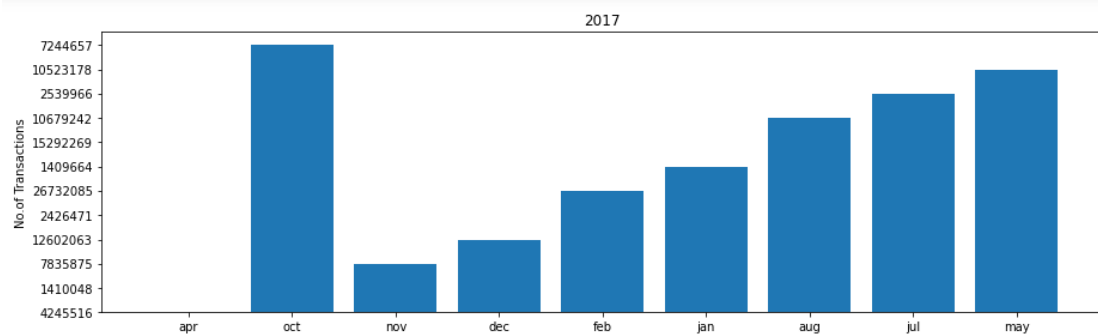
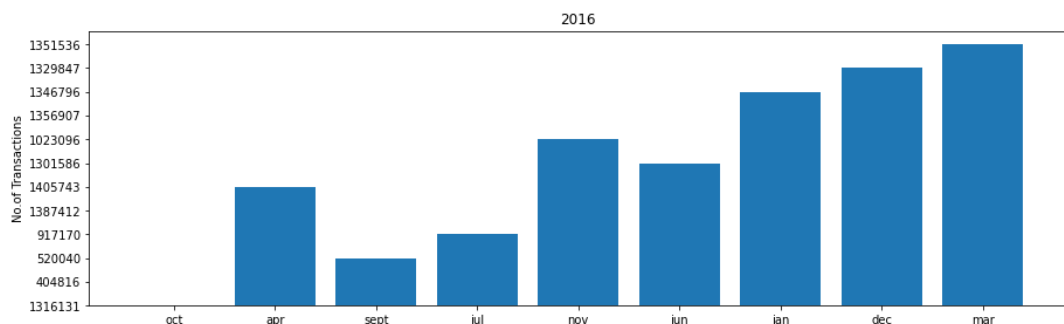
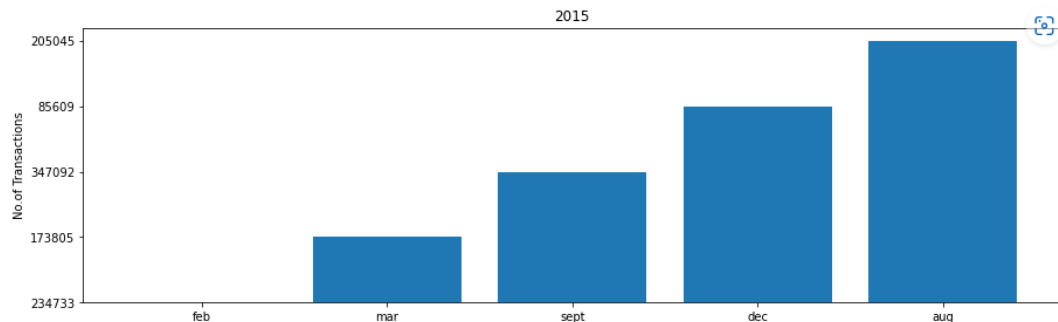
BIG DATA PROCESSING_ECS765P_COURSEWORK_REPORT

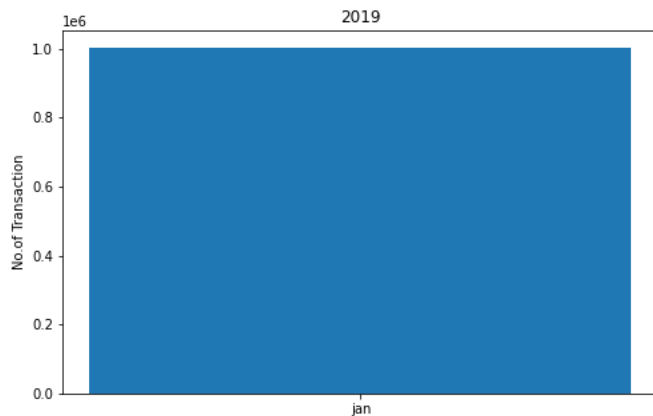
Student Number: 220431413

Name: Jude Damian Sequeira

A. PART A(Number of Transactions):

Text(0, 0.5, 'No.of Transactions')





The above bar plots are of the number of Transactions in each month for each year from 2015-2019. For this task, the Transactions data of ethereum-parvulus was used and the field “block_timestamp” was preprocessed to obtain the month-Year mapping followed by a reducer to obtain the aggregated output of each date registered.

Below is the main part of the code from gApart1.py file

Code:

```
lines = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-
parvulus/transactions.csv")

clean_lines = lines.filter(good_line)

d = clean_lines.map(lambda b: (datetime.fromtimestamp(int(b.split(',')[11])).strftime("%m-%Y"),
1))

dates = d.reduceByKey(operator.add)
```

The output was stored as a text file (i.e partAoutput.txt) with list of all the dates in month-year format along with its total number of transactions.

Plot Analysis(PartA 1):

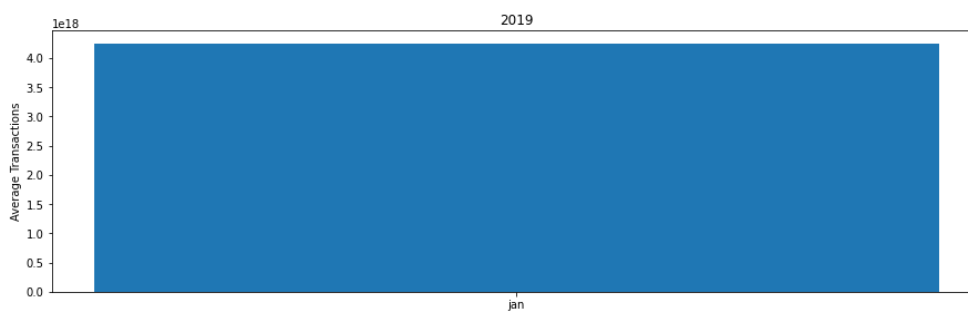
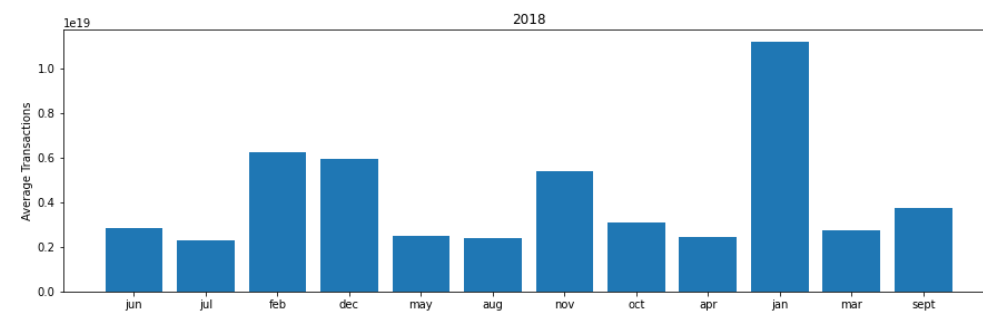
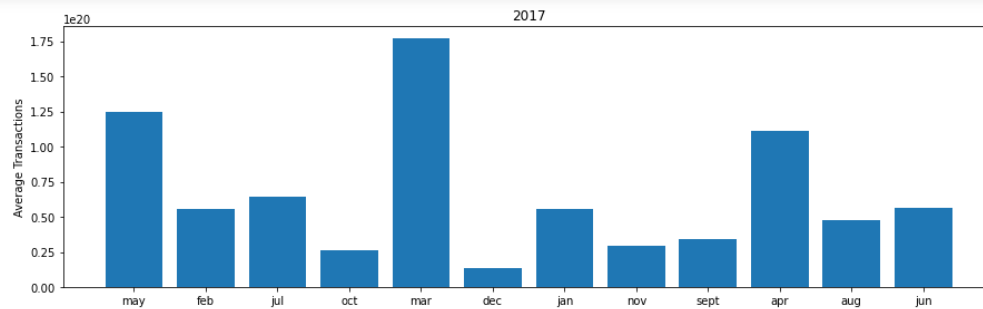
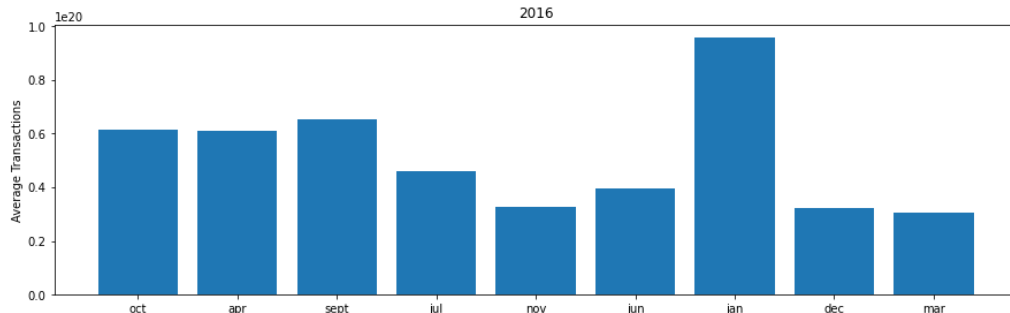
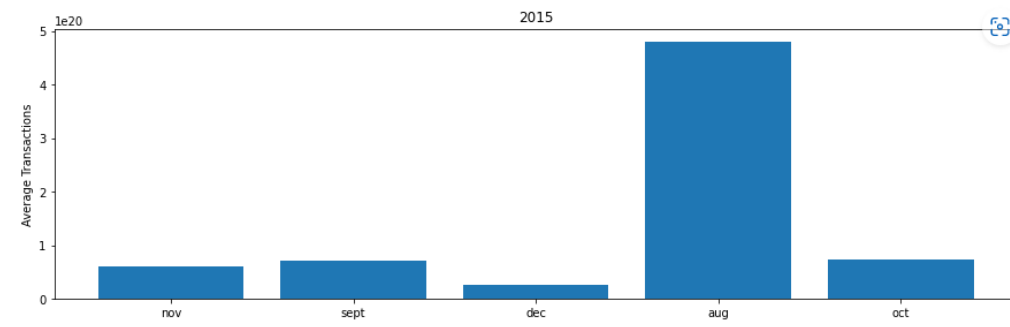
The bar plot obtained was using matplotlib library and basic preprocessing of the output text file to obtain a dataframe which was used for the bar plot. These plots are in the “*partA_plots.ipyn*” file.

From analysing the bar plots, we can observe that during 2015 the highest number of transactions occurred during the month of august, during 2016 it was march, followed by October in 2017 and June in 2018. During 2019, only the month of January timestamps were recorded with 1002431 transactions.

Overall, the number of transactions for each month keep on increasing from year 2015 to 2018. The month of March has seen lowest transactions during 2015 as well as 2018, highest being during 2016. The highest number of transactions was recorded in June 2018(i.e 16056742 transactions) and the lowest was during march 2015(i.e 173805 transactions).

B. PART A(Average Number of Transactions):

Text(0, 0.5, 'Average Transactions')



The above plot is used to analyse the Average Value of Transaction per month for each year. The procedure is similar to the previous section but with an extra field “value” of transaction data included.

The code below converts the block_timestamps to month-year format as well as preprocesses the “Value” field of transaction, followed by summing all values of same date with count using reducer and finally mapping all average values with its respective date value.

Below is the main part of the code from gApart2.py file

Code:

```
clean_lines = lines.filter(good_line)

val = clean_lines.map(lambda b:((datetime.fromtimestamp(int(b.split(',')[11])).strftime("%m-%Y") ,
(int(b.split(',')[7]), 1)))) # adds the two entries of same date

red = val.reduceByKey(lambda x, y: (x[0]+y[0], x[1]+y[1])) # summation of all values of same
date with count

avg_tr = red.map(lambda a: (a[0], str(a[1][0]/a[1][1]))) # mapping of average values

avg_tr = avg_tr.map(lambda z: ', '.join(str(t) for t in z))
```

The output was stored as a text file (i.e partA_P2.txt) with list of all the dates in month-year format along with its average value of transactions.

Plot Analysis(PartA_2):

The bar plot obtained was using matplotlib library and basic preprocessing of the output text file to obtain a dataframe which was used for the bar plot. These plots are in the “*partA_plots.ipyn*” file.

From analysing the bar plots, we can observe that during 2015 the highest average transaction value occurred during the month of august, during 2016 and 2018 it was January, followed by march in 2017. During 2019, only the month of January average values were recorded.

Overall, the highest average value of transactions was recorded in march 2015 with around 5.0E20 value although the other months have less than 1.0E20 for that particular year. July 2018 has recorded the least value in terms of average with value around 0.2E19.

C. PART B (Top 10 Most Popular Services):

Below is the main part of the code from qB.py file

CODE:

```
lines_tr = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-parvulus/transactions.csv")

lines_con = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket +
"/ECS765/ethereum-parvulus/contracts.csv")

tr_clean_lines = lines_tr.filter(good_line)

con_clean_lines = lines_con.filter(good_line_con)

trans = tr_clean_lines.map(lambda x: (x.split(',')[6], int(x.split(',')[7])))

cont = con_clean_lines.map(lambda x: (x.split(',')[0],1))

transaction = trans.reduceByKey(lambda x, y: x + y)

join_df = transaction.join(cont)      #join transaction adress field to contract tables to_adress field

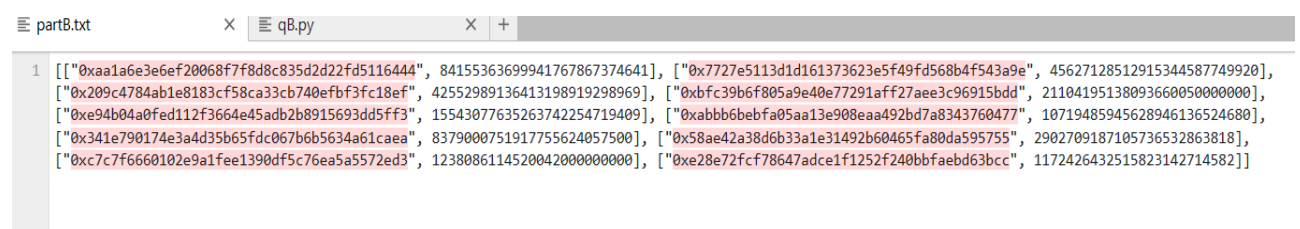
address_value=join_df.map(lambda x: (x[0], x[1][0]))

top10 = address_value.takeOrdered(10, key=lambda x: -1*x[1])
```

The next analysis finds the top 10 smart contracts by total ether received by joining the 'address' field of contracts dataset to the 'to_address' from transactions data. The above code preprocesses the text lines for both datasets separately. The 'to_address' and 'value' fields from transaction dataset and the 'address' (Address of the Contract) field from contract fields are considered. Both the address fields are joined using the *join()* function followed by mapping of the addresses along with the amount of ether received per address of the contract in descending order using the *takeOrdered()* function.

The output of the spark job is stored in a text file named 'partB.txt', which lists the top 10 smart contracts in descending order. The following address is the highest of the 10 smart contracts: `["0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444", 84155363699941767867374641]`

Output



```
1  [{"0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444", 84155363699941767867374641}, {"0x7727e5113d1d161373623e5f49fd568b4f543a9e", 45627128512915344587749920}, {"0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef", 42552989136413198919298969}, {"0xbfc39b6f805a9e40e77291aff27aee3c96915bdd", 2110419513809366005000000}, {"0xe94b04a0fed112f3664e45adb2b8915693dd5ff3", 15543077635263742254719409}, {"0xabbb6bebf805aa13e908aaa492bd7a8343760477", 10719485945628946136524680}, {"0x341e790174e3a4d35b65fdc067b6b5634a61caea", 8379000751917755624057500}, {"0x58ae42a38d6b33a1e31492b60465fa80da595755", 2902709187105736532863818}, {"0xc7c7f6660102e9a1fee1390df5c76ea5a5572ed3", 1238086114520042000000000}, {"0xe28e72fcf78647adce1f1252f240bbfaebd63bcc", 1172426432515823142714582}]
```

D. PART C (Top 10 Most Active Miners):

Below is the main part of the code from qC.py file

CODE:

```
lines = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-
parvulus/blocks.csv")

clean_lines = lines.filter(good_line)

blocks = clean_lines.map(lambda b: (str(b.split(',')[9]),int(b.split(',')[12])))    #mapping miner and
size fields

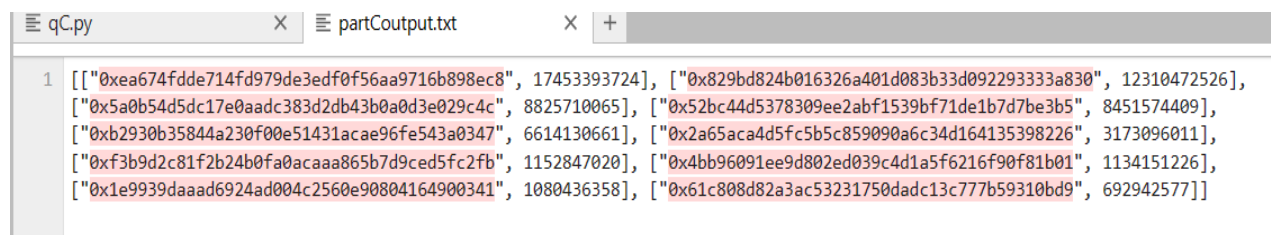
block_new = blocks.reduceByKey(operator.add)

top_10 = block_new.takeOrdered(10, key = lambda x: -1*x[1])    # getting top10 miners in terms of
size from descending ordered list
```

The next analysis finds the top 10 most active miners in terms of blocks mined, by using the the 'miner' and 'size' fields of the *blocks* dataset. The above code pre-processes the text lines as usual and maps the mentioned fields followed by reducer operation to obtain a list of miner addresses along with the size of the blocks mined by each. The takeOrdered() function finds the top 10 miners with the blocks mined and stores the output in a text file named 'partCoutput.txt'. The miner with the highest size of blocks mined is

```
["0xea674fdde714fd979de3edf0f56aa9716b898ec8", 17453393724]
```

Output:



```
1 [{"0xea674fdde714fd979de3edf0f56aa9716b898ec8", 17453393724}, {"0x829bd824b016326a401d083b33d092293333a830", 12310472526}, {"0x5a0b54d5dc17e0aad383d2db43b0a0d3e029c4c", 8825710065}, {"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5", 8451574409}, {"0xb2930b35844a230f00e51431acae96fe543a0347", 6614130661}, {"0x2a65aca4d5fc5b5c859090a6c34d164135398226", 3173096011}, {"0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb", 1152847020}, {"0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01", 1134151226}, {"0x1e9939daaad6924ad004c2560e90804164900341", 1080436358}, {"0x61c808d82a3ac53231750dad13c777b59310bd9", 692942577}]
```

