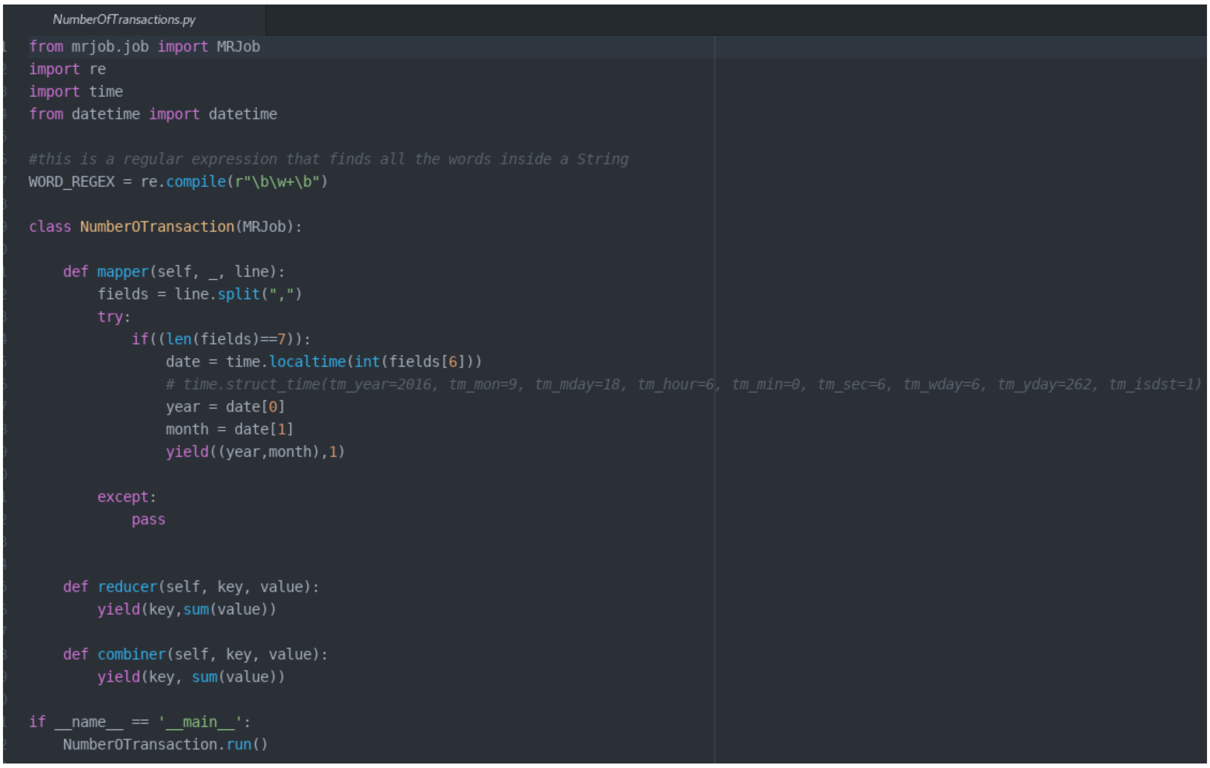
COURSEWORK: ETHEREUM ANALYSIS (20%)

-HARNOOR SINGH OBERAI

190753898

Assignment submitted on 2nd December. Hence eligible for extra 5 marks.

PART A. TIME ANALYSIS (30%)



```python
from mrjob.job import MRJob
import re
import time
from datetime import datetime

#this is a regular expression that finds all the words inside a String
WORD_REGEX = re.compile(r"\b\w+\b")

class NumberOTransaction(MRJob):

    def mapper(self, _, line):
        fields = line.split(",")
        try:
            if((len(fields)==7)):
                date = time.localtime(int(fields[6]))
                # time.struct_time(tm_year=2016, tm_mon=9, tm_mday=18, tm_hour=6, tm_min=0, tm_sec=6, tm_wday=6, tm_yday=262, tm_isdst=1)
                year = date[0]
                month = date[1]
                yield((year,month),1)

        except:
            pass


    def reducer(self, key, value):
        yield(key,sum(value))

    def combiner(self, key, value):
        yield(key, sum(value))

if __name__ == '__main__':
    NumberOTransaction.run()
```

JOB ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1574975221160_5900/

Hadoop commad =
bash-4.2$ python NumberOfTransactions.py -r hadoop --output-dir PartA --no-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

Mapper:

1. All the transaction file in /data/ethereum/transaction will be the input file
2. Converting the date field in readable format of year and date which will act as key and value will be set to 1.

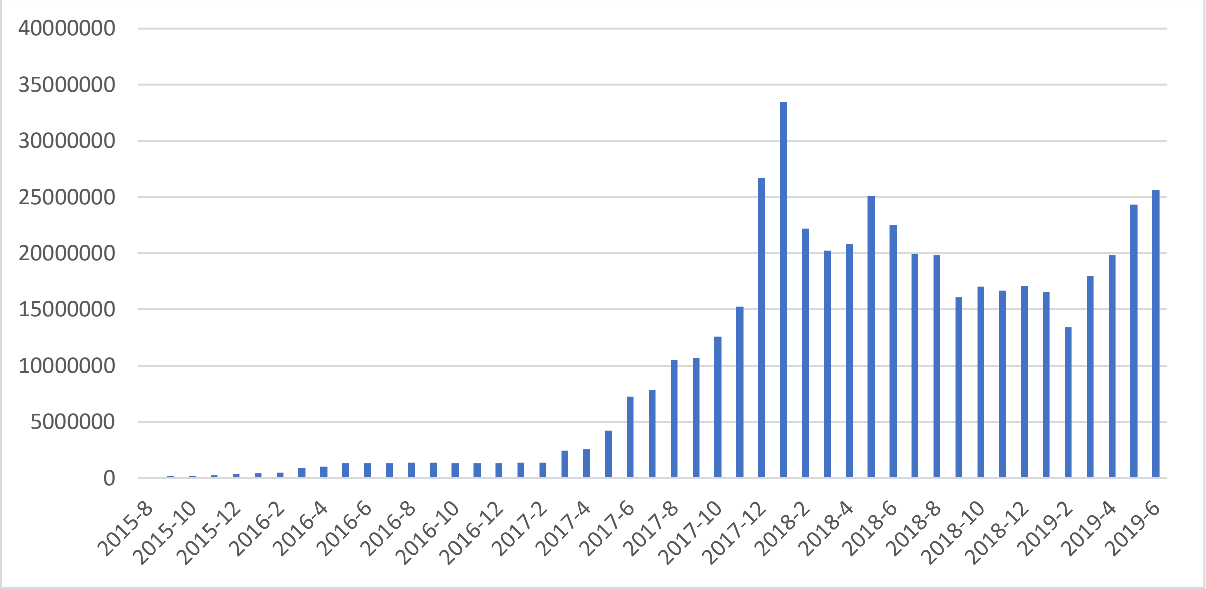
Reducer:

1. Reducer will calculate the total number of values passed in a given key(year,month) which in turn will yield the total number of transaction in a given year and month.

Combiner:

1. Combiner is added to increase the efficiency of code after mapper.

The following bar plot shows number of transaction occurring in a given Year-Month.

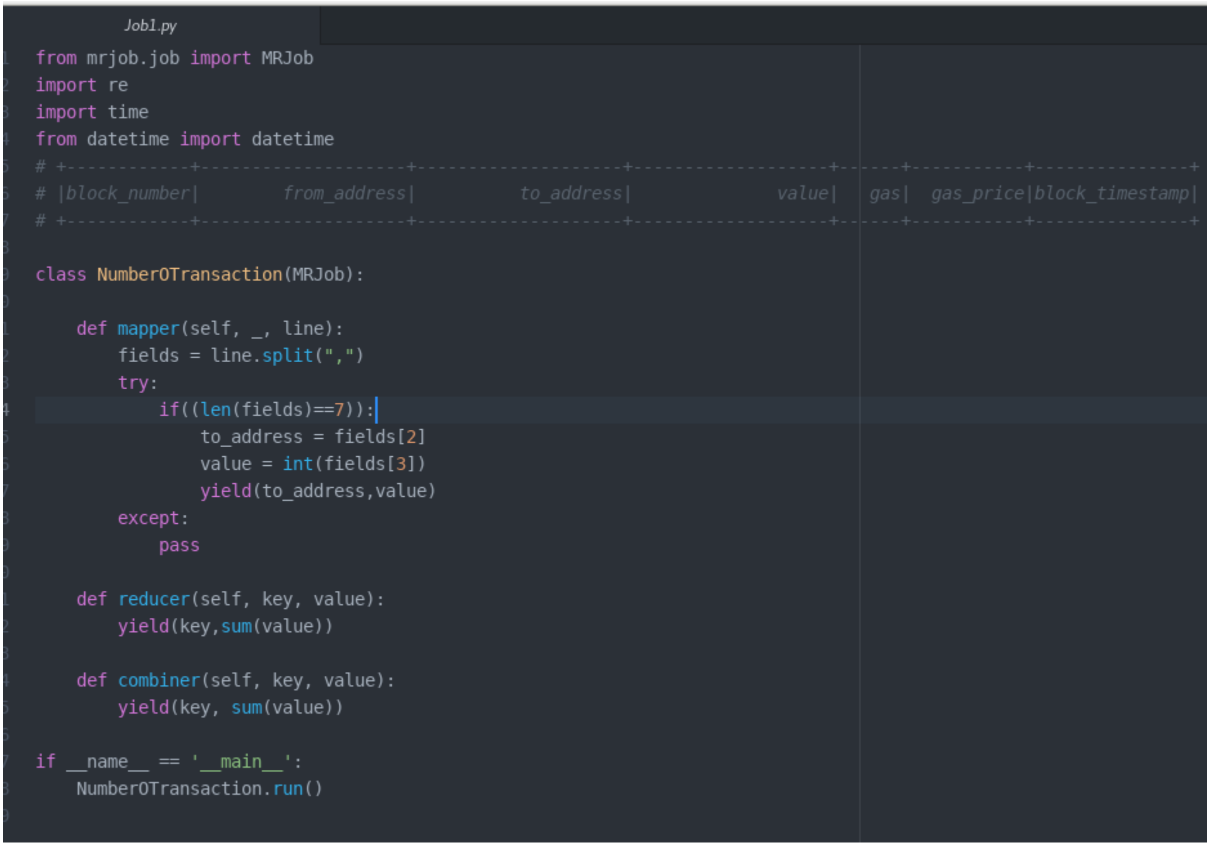


Y-axis represent = YEAR-MONTH
X-axis represent = Number of Transaction.

Observation:
Maximum number of transaction occurred in 2018-1
Minimum number of transaction occurred in 2015-8

Part B. Top Ten Most Popular Services (40%)

Job 1 - Initial Aggregation



```python
from mrjob.job import MRJob
import re
import time
from datetime import datetime
# +------------+--------------------+--------------------+-------------------+------+-----------+---------------+
# |block_number|        from_address|          to_address|              value|   gas|  gas_price|block_timestamp|
# +------------+--------------------+--------------------+-------------------+------+-----------+---------------+

class NumberOTransaction(MRJob):

    def mapper(self, _, line):
        fields = line.split(",")
        try:
            if((len(fields)==7)):
                to_address = fields[2]
                value = int(fields[3])
                yield(to_address,value)
        except:
            pass

    def reducer(self, key, value):
        yield(key,sum(value))

    def combiner(self, key, value):
        yield(key, sum(value))

if __name__ == '__main__':
    NumberOTransaction.run()
```

Job ID :
http://andromeda.student.eecs.qmul.ac.uk:19888/jobhistory/job/job_1574171293853_2421

Hadoop commad =
bash-4.2$ python Job1.py -r hadoop --output-dir outPart_B_job1 --no-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

Mapper:

1. All the transaction file in /data/ethereum/transaction will be the input file
2. We will use to_address as key and the value as “value” from transaction table.

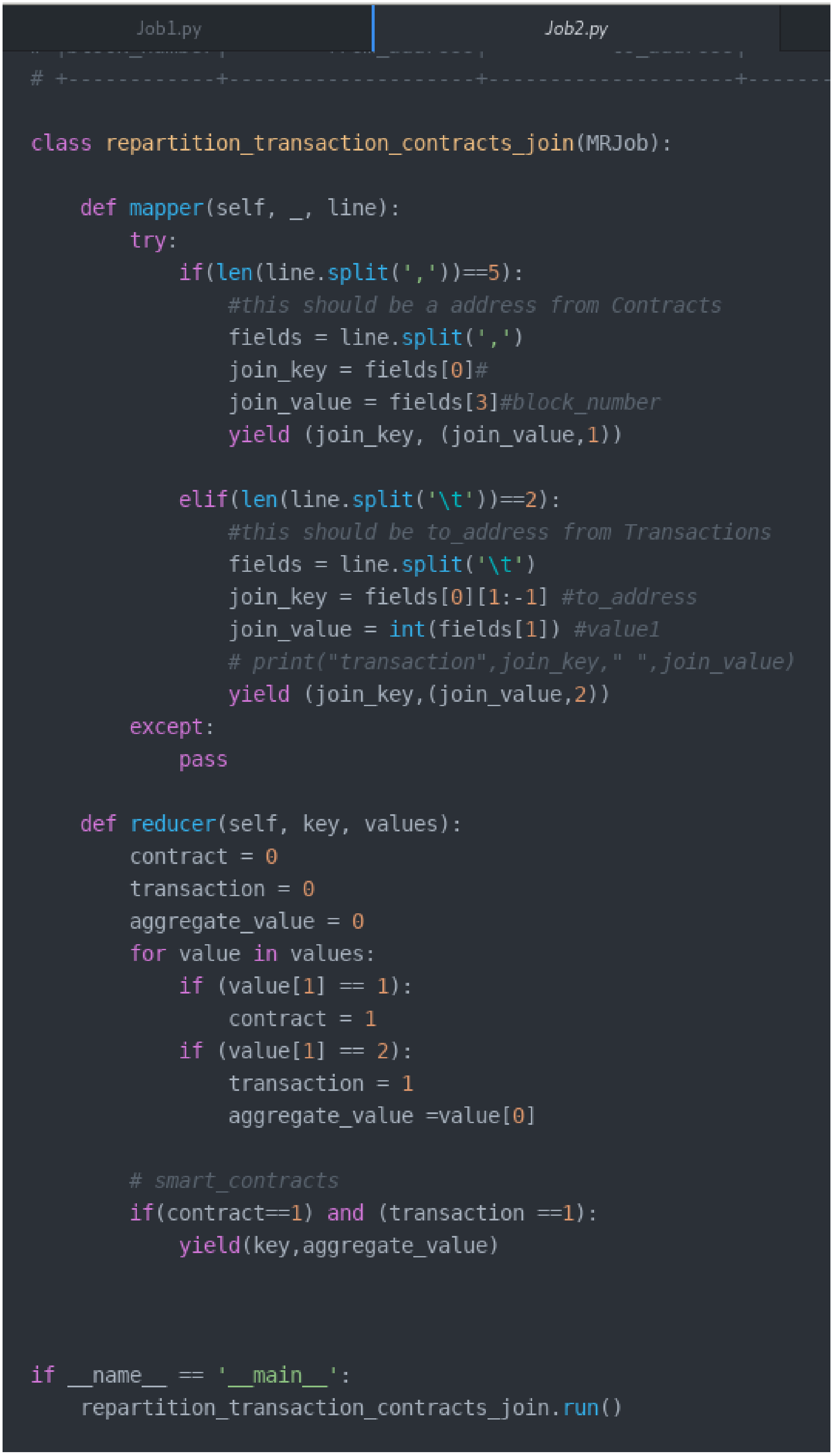
Reducer:

1. Reducer will calculate the total number of values passed in a given key(to_address) which in turn will yield the total number of values in a given address.

Combiner:

1. Combiner is added to increase the efficiency of code after mapper.

Job 2 - Joining transactions/contracts and filtering



```
# +------------+-------------------+-------------------+------

class repartition_transaction_contracts_join(MRJob):

    def mapper(self, _, line):
        try:
            if(len(line.split(','))==5):
                #this should be a address from Contracts
                fields = line.split(',')
                join_key = fields[0]#
                join_value = fields[3]#block_number
                yield (join_key, (join_value,1))

            elif(len(line.split('\t'))==2):
                #this should be to_address from Transactions
                fields = line.split('\t')
                join_key = fields[0][1:-1] #to_address
                join_value = int(fields[1]) #value1
                # print("transaction",join_key," ",join_value)
                yield (join_key,(join_value,2))
        except:
            pass

    def reducer(self, key, values):
        contract = 0
        transaction = 0
        aggregate_value = 0
        for value in values:
            if (value[1] == 1):
                contract = 1
            if (value[1] == 2):
                transaction = 1
                aggregate_value =value[0]

        # smart_contracts
        if(contract==1) and (transaction ==1):
            yield(key,aggregate_value)



if __name__ == '__main__':
    repartition_transaction_contracts_join.run()
```

JobID:
http://andromeda.student.eecs.qmul.ac.uk:19888/jobhistory/job/job_1574171293853_4647

Hadoop command :

```
python Job2.py -r hadoop hdfs://andromeda.eecs.qmul.ac.uk/user/hso30/outPart_B_job1 --output-dir outPart_B_job2 --no-output
hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/contracts
```

Repartition join is used since we are joining two very large datasets. Following files are used:

1. All files from /data/ethereum/contracts
2. All files from PartBJob1 which I have stored in hdfs://andromeda.eecs.qmul.ac.uk/user/hso30/outPart_B_job1

Mapper:

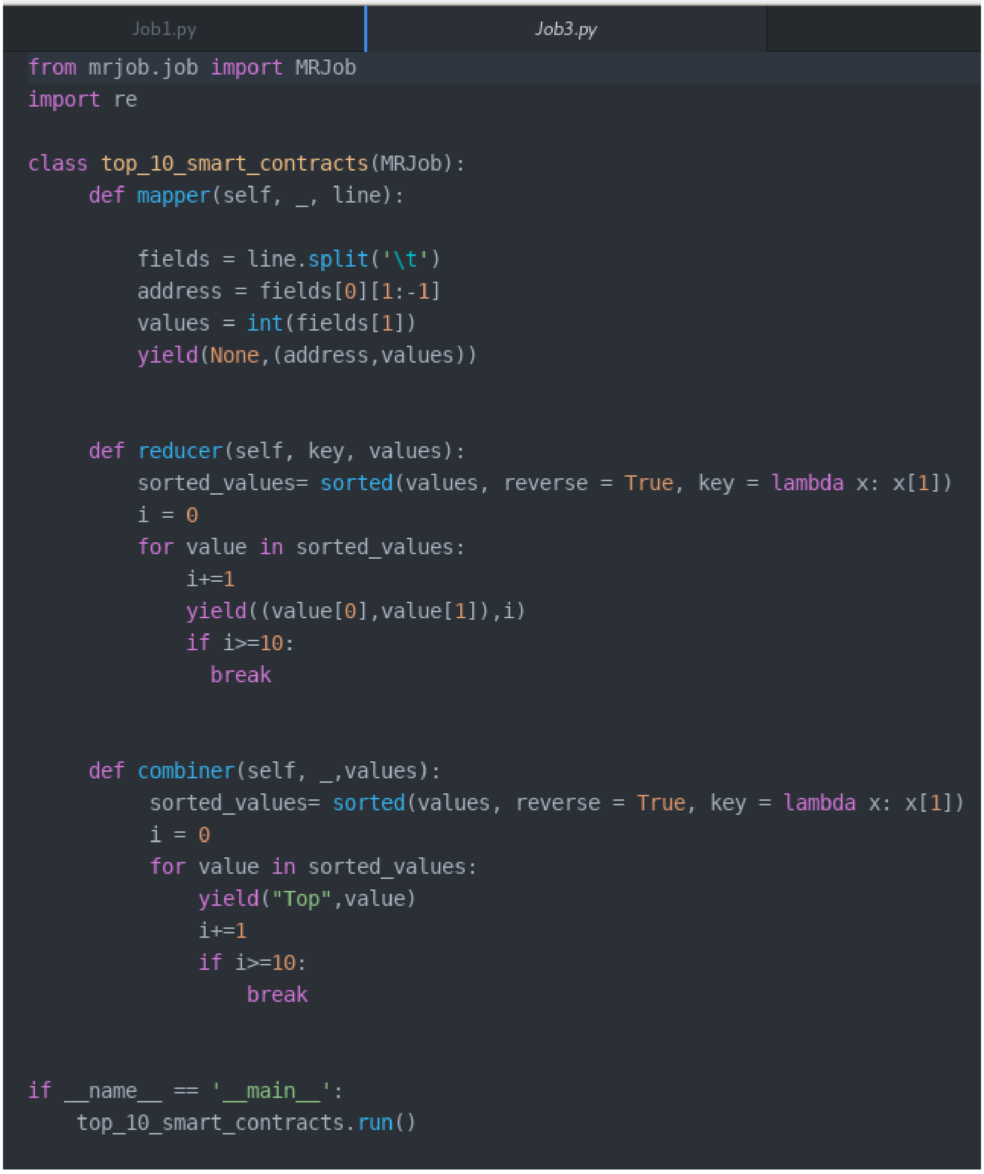
1. Mapper will differentiate whether which file is coming as input by using if statement to check. If splitting the line by "," have number of field as 5, the file is from contracts. If splitting the line by "\t" have number of field as 2, the file is from Job1.
2. Join is performed using address field in both contracts and Job1.
3. Files from contracts:
   a) key = address
   b) value = (block_number, 1). Value[1] indicates that mapper is using contract file which will be used in reducer.
4. Files from Job1:
   a) key = address
   b) value = (aggregate_total, 2). Value[0] is the total sum obtained from Job1 and 2 that mapper is using Job1 file which will be used in reducer.

Reducer:

1. Reducer distinguishes whether we are having a smart contract or user contract.
2. A smart contract occurs if the address appears in both contract and Job1 file.
3. I am using two variable contracts and aggregation to check if both the files have same address.
4. For smart contract:
   a) Input for reducer: (address, ((block_number,1),(aggregate_value,2))).
   b) Since we have both contract and job1 value, we set contract and transaction variable to 1.
   c) if(contract==1) and (transaction ==1): yield (address,aggreate_value)
5. For user Contract:
   a) Input for reducer: (address, (block_number,1)).
   b) We set contract variable to 1. Transaction to 0.
   c) if(contract==1) and (transaction ==1): // our code won't enter this statement and will not yield anything omitting out user contract.

I have not uploaded the output for Job2(126 MB) in zip folder due to space constraint. If required additional info, I have uploaded the file in my local drive.

Job 3 - Top Ten



```
from mrjob.job import MRJob
import re

class top_10_smart_contracts(MRJob):
     def mapper(self, _, line):

         fields = line.split('\t')
         address = fields[0][1:-1]
         values = int(fields[1])
         yield(None,(address,values))


     def reducer(self, key, values):
         sorted_values= sorted(values, reverse = True, key = lambda x: x[1])
         i = 0
         for value in sorted_values:
             i+=1
             yield((value[0],value[1]),i)
             if i>=10:
               break


     def combiner(self, _,values):
          sorted_values= sorted(values, reverse = True, key = lambda x: x[1])
          i = 0
          for value in sorted_values:
              yield("Top",value)
              i+=1
              if i>=10:
                  break


if __name__ == '__main__':
    top_10_smart_contracts.run()
```

Command :

Using the output file created from Job2 run the following command in python

python Job3.py Job2Output.txt > outputTop10.txt

Sample input file :
"a" 25
"b" 43
"c" 2
Mapper:

1. Splitting line by tab.
2. Address field removes “” by using fields[0][1:-1].
3. Key = none // to make every key same making reducer calculate top 10.
4. Value = (address, value)

Reducer:

1. We sort the value in descending order.
2. After sorting print the top 10 values.

Combiner:

1. Increase the efficiency of our map/reduce job. It summarize the map output records with the same key.

Top 10 addresses:

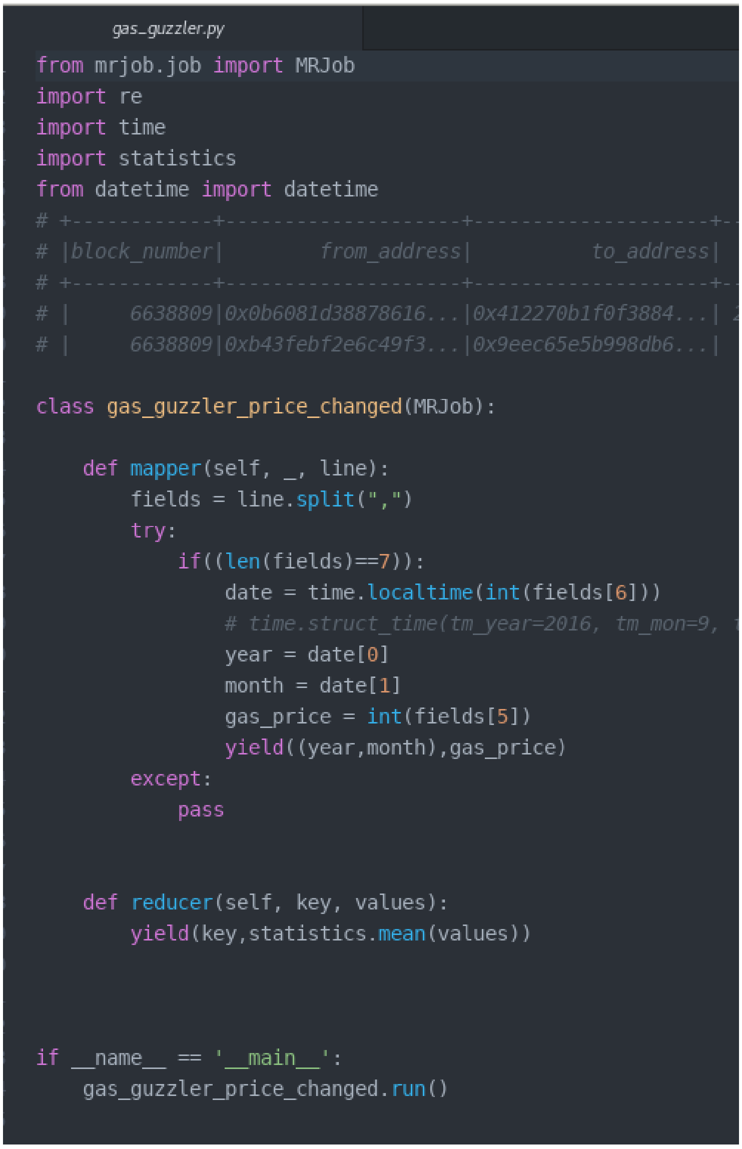
- ["0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444", 84155100809965865822726776] 1
- ["0xfa52274dd61e1643d2205169732f29114bc240b3", 45787484483189352986478805] 2
- ["0x7727e5113d1d161373623e5f49fd568b4f543a9e", 45620624001350712557268573] 3
- ["0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef", 43170356092262468919298969] 4
- ["0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8", 27068921582019542499882877] 5
- ["0xbfc39b6f805a9e40e77291aff27aee3c96915bdd", 21104195138093660050000000] 6
- ["0xe94b04a0fed112f3664e45adb2b8915693dd5ff3", 15562398956802112254719409] 7
- ["0xbb9bc244d798123fde783fcc1c72d3bb8c189413", 11983608729202893846818681] 8
- ["0xabbb6bebfa05aa13e908eaa492bd7a8343760477", 11706457177940895521770404] 9
- ["0x341e790174e3a4d35b65fdc067b6b5634a61caea", 8379000751917755624057500]   10

Part C. Data exploration (30%)

Miscellaneous Analysis

1. Gas Guzzlers (15/30)

(A) How has gas price changed over time?



```python
from mrjob.job import MRJob
import re
import time
import statistics
from datetime import datetime
# +------------+--------------------+--------------------+-
# |block_number|        from_address|          to_address|
# +------------+--------------------+--------------------+-
# |     6638809|0x0b6081d38878616...|0x412270b1f0f3884...|
# |     6638809|0xb43febf2e6c49f3...|0x9eec65e5b998db6...|

class gas_guzzler_price_changed(MRJob):

    def mapper(self, _, line):
        fields = line.split(",")
        try:
            if((len(fields)==7)):
                date = time.localtime(int(fields[6]))
                # time.struct_time(tm_year=2016, tm_mon=9,
                year = date[0]
                month = date[1]
                gas_price = int(fields[5])
                yield((year,month),gas_price)
        except:
            pass


    def reducer(self, key, values):
        yield(key,statistics.mean(values))



if __name__ == '__main__':
    gas_guzzler_price_changed.run()
```

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1574975221160_0519/

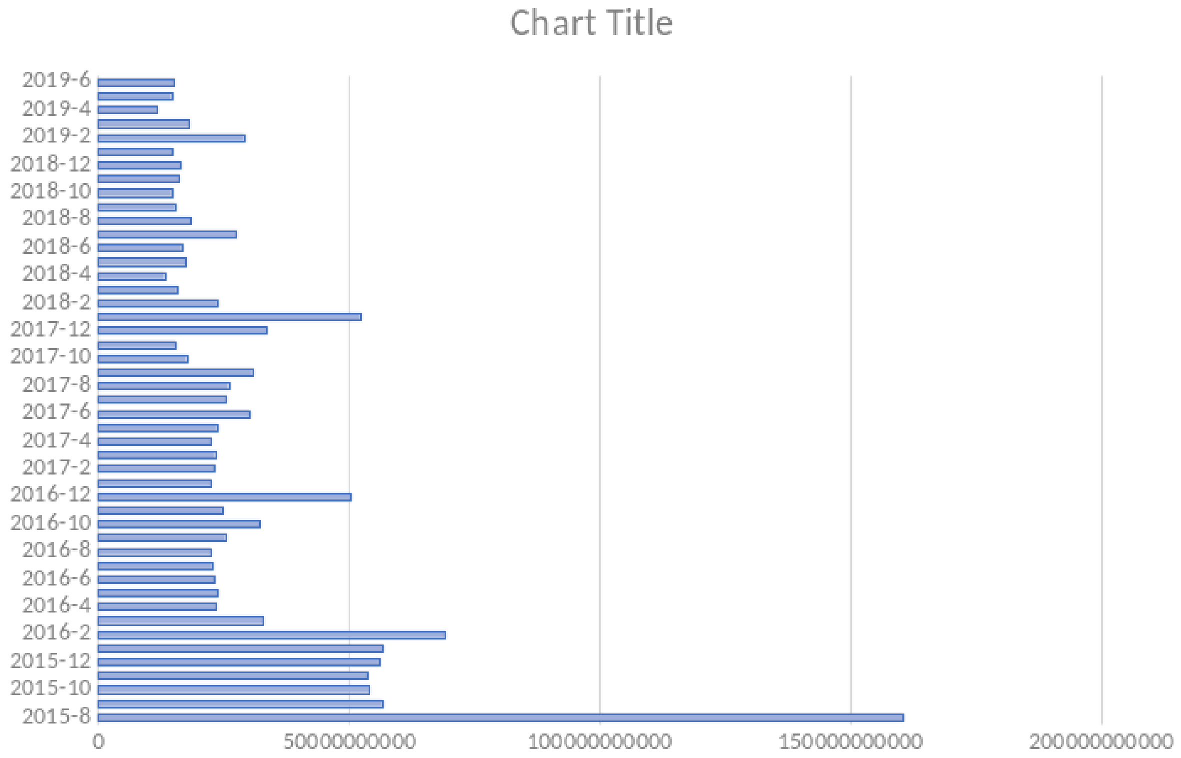
Hadoop command:
python gas_guzzler.py -r hadoop --output-dir outPartC_gasGuzzler --no-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

Mapper:

1. We input all the transaction files in /data/ethereum/transactions.
2. Converting the date into readable format of year month.
3. Key = (year, month)
4. value = gas price

Reducer:

1. yield(key,statistics.mean(values))
2. In this function calculate the average value occurring in particular year and month.



Y-axis represent = YEAR-MONTH

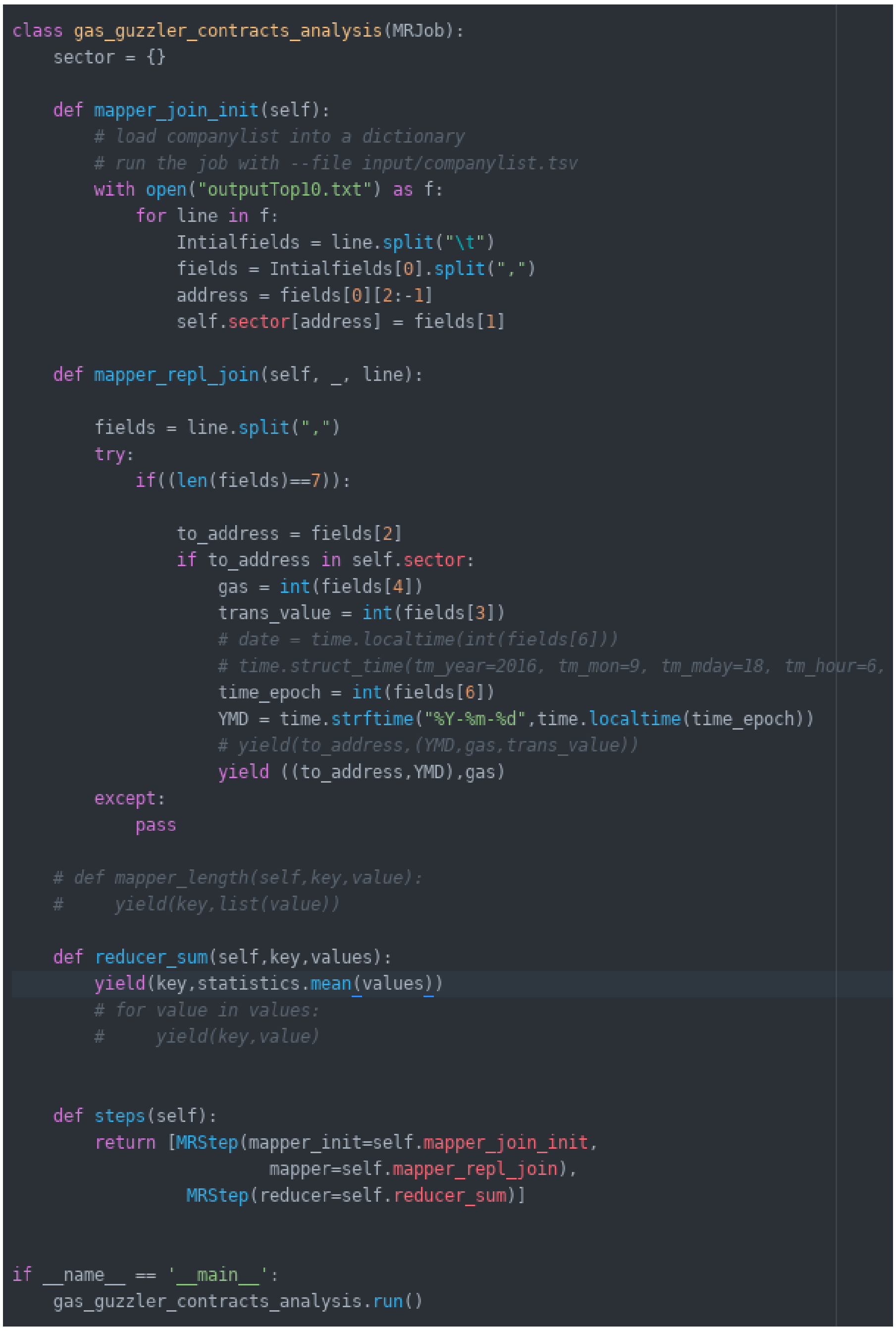
X-axis represent = Mean average of gas values

Observation:

Maximum Mean recorded in 2015-8

Minimum Mean recorded in 2019-4

(B) Have contracts become more complicated, requiring more gas, or less so? How does this correlate with your results seen within Part B.



```
class gas_guzzler_contracts_analysis(MRJob):
    sector = {}

    def mapper_join_init(self):
        # load companylist into a dictionary
        # run the job with --file input/companylist.tsv
        with open("outputTop10.txt") as f:
            for line in f:
                Intialfields = line.split("\t")
                fields = Intialfields[0].split(",")
                address = fields[0][2:-1]
                self.sector[address] = fields[1]

    def mapper_repl_join(self, _, line):

        fields = line.split(",")
        try:
            if((len(fields)==7)):

                to_address = fields[2]
                if to_address in self.sector:
                    gas = int(fields[4])
                    trans_value = int(fields[3])
                    # date = time.localtime(int(fields[6]))
                    # time.struct_time(tm_year=2016, tm_mon=9, tm_mday=18, tm_hour=6,
                    time_epoch = int(fields[6])
                    YMD = time.strftime("%Y-%m-%d",time.localtime(time_epoch))
                    # yield(to_address,(YMD,gas,trans_value))
                    yield ((to_address,YMD),gas)
        except:
            pass

    # def mapper_length(self,key,value):
    #     yield(key,list(value))

    def reducer_sum(self,key,values):
        yield(key,statistics.mean(values))
        # for value in values:
        #     yield(key,value)


    def steps(self):
        return [MRStep(mapper_init=self.mapper_join_init,
                         mapper=self.mapper_repl_join),
                 MRStep(reducer=self.reducer_sum)]


if __name__ == '__main__':
    gas_guzzler_contracts_analysis.run()
```

JobID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1574975221160_6488/

Input : top 10 contract from PARTB_job3
Hadoop command:
python gas_guzzler_contracts_analysis.py -r hadoop --file outputTop10.txt --output-dir partC_contractAnalysis --no-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

We perform a replication join on top 10 contracts and the transaction files
Initial steps defined to execute in order:

```
def steps(self):
    return [MRStep(mapper_init=self.mapper_join_init,
                mapper=self.mapper_repl_join),
          MRStep(mapper=self.mapper_length,
              reducer=self.reducer_sum)]
```

Mapper:

1. mapper_join_init(self)
   a) store the data from top 10 contracts in sector mapper as sector[key] = value where key is the address of contract and value is the total aggregate obtained in top 10 contract file.
2. mapper_repl_join(self, _, line)
   a) All the transaction files are used here.
   b) We calculate date in readable format of Year-Month-date.
   c) yield ((to_address,YMD),gas) where key =(address,YMD) and values = (gas)

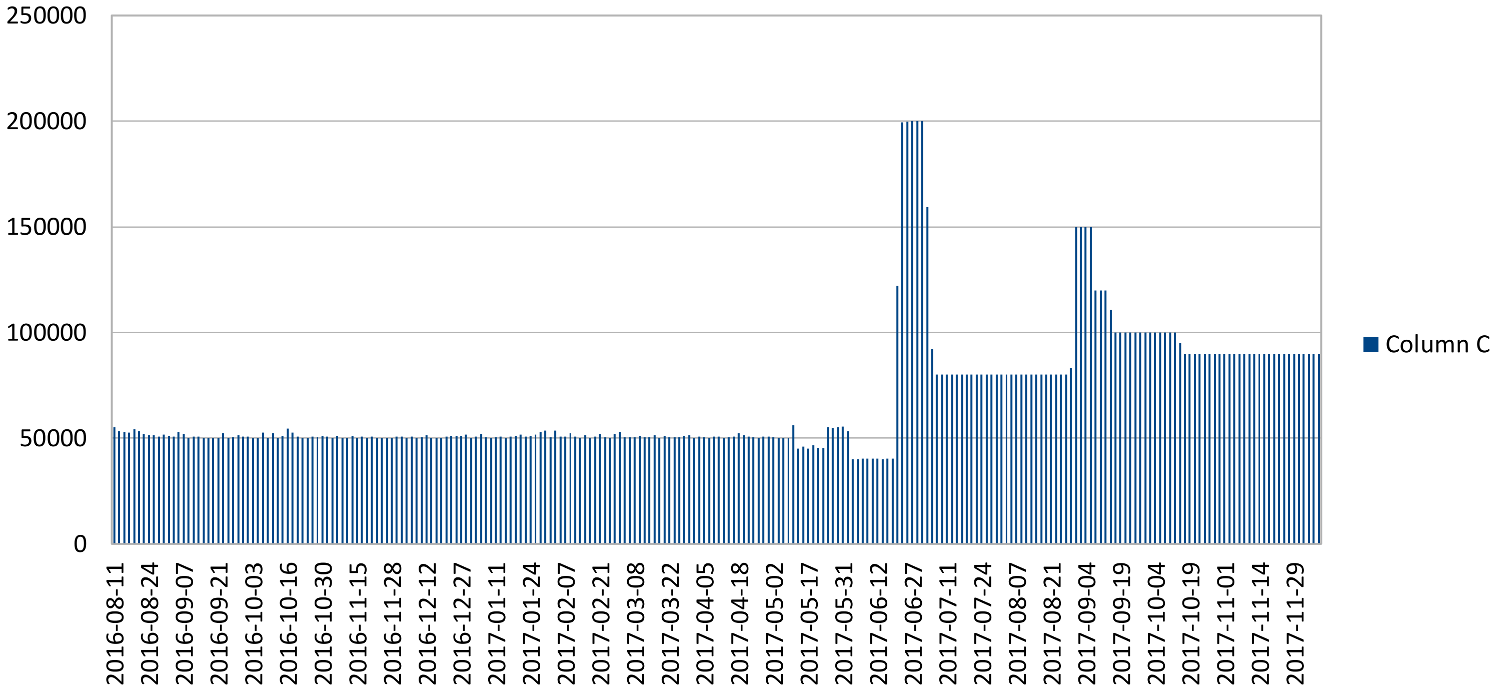
Reducer:

1. The reducer is getting the mean of gas for YEAR-MONTH-DATE.

I have done the analysis for top 3 smart contract's address from PART_B Job3:

X-axis: Date in Year-Month-Date format
Y-axis: Gas Value

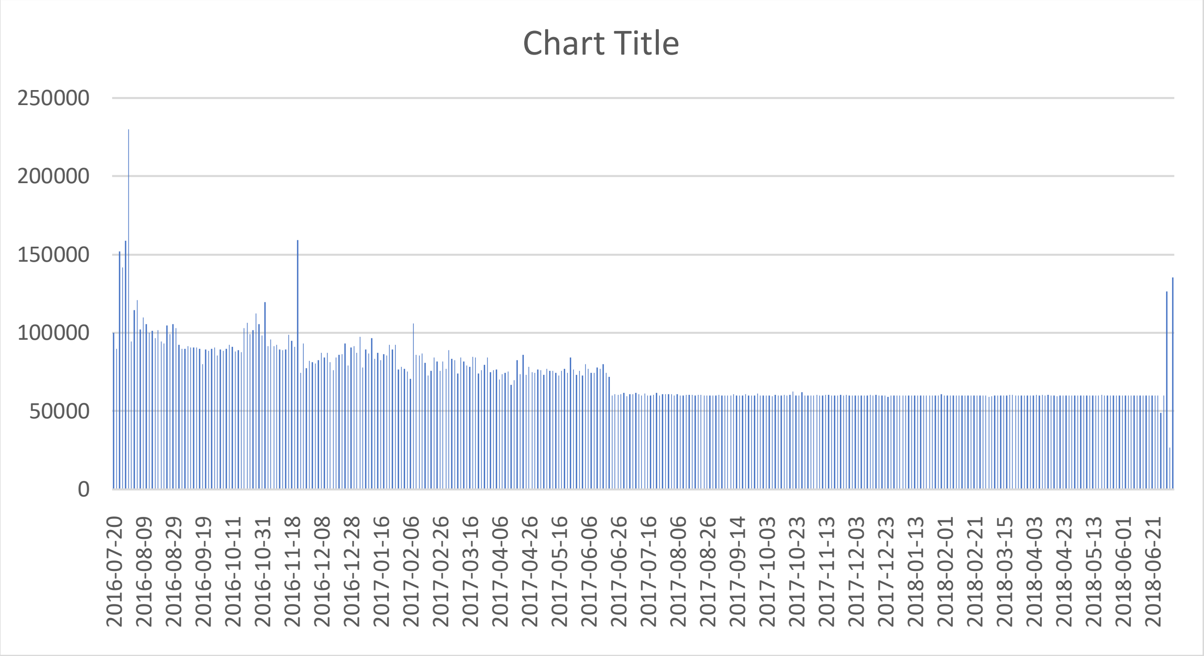
(I) Address: 0x7727e5113d1d161373623e5f49fd568b4f543a9e



Observation:

The most common gas values for the given address is around 50,000. We Can see an increase in 2017-06-12 gas value. Hence the cost to mine ether increased in 2017-June. The contacts have become more complicated after 2017-06 as the gas has increased.

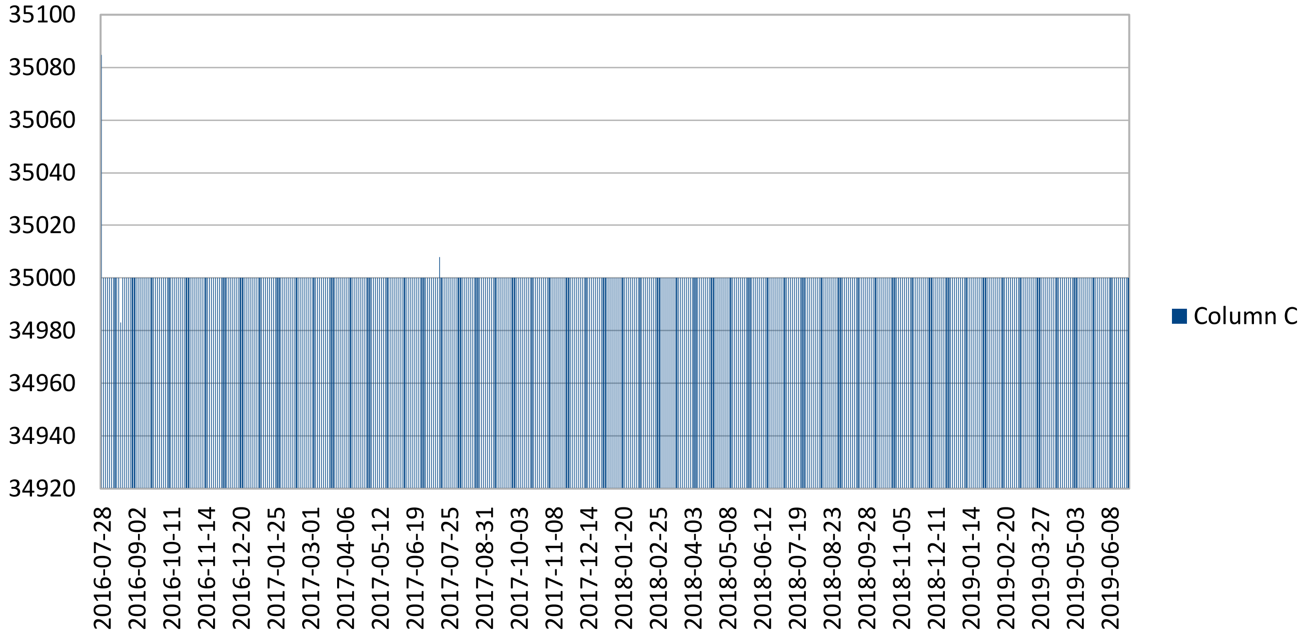
(II) 0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444



Observation:

There are spikes observed for this address for the year 2016-07 to 2017-06. After which gas value remains kind of constant at 50,000. The contracts have become simpler and stable .

(III) 0xfa52274dd61e1643d2205169732f29114bc240b3

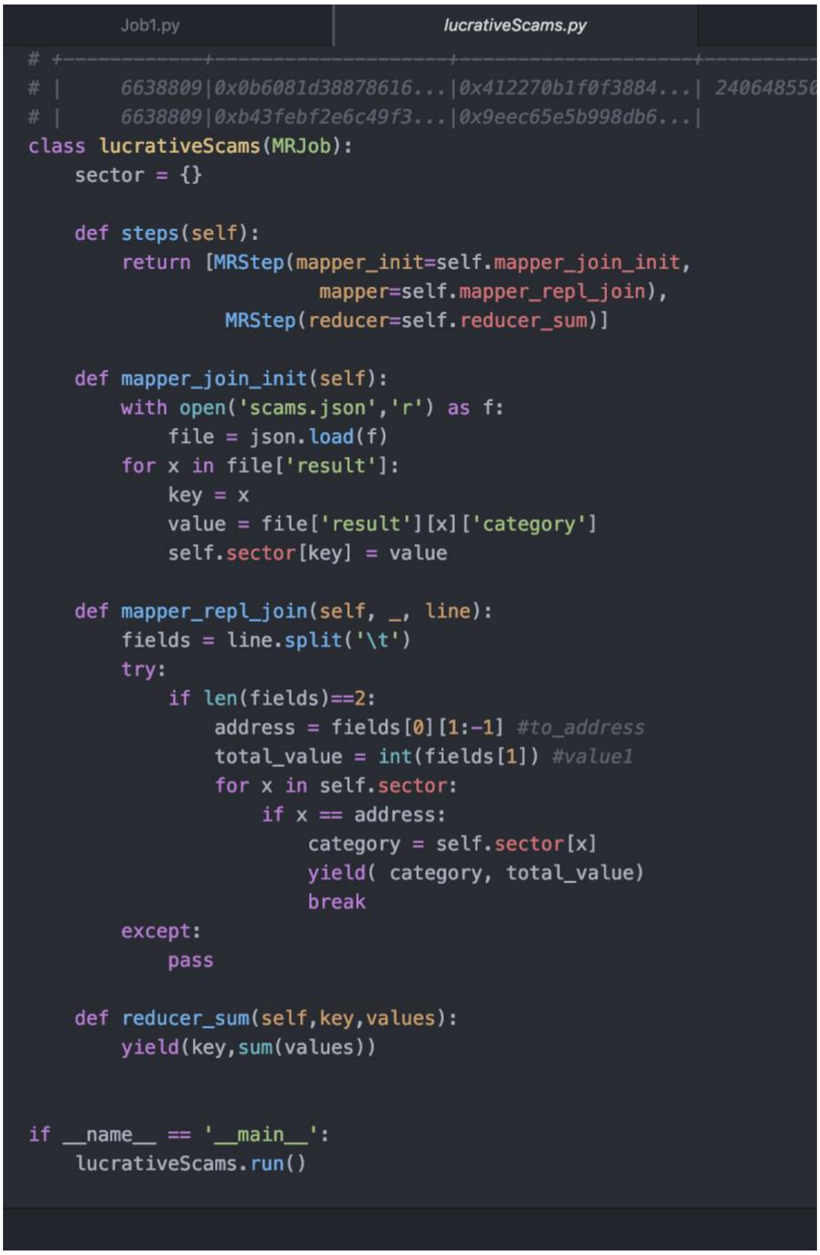


Observation :

Gas value remains constant most of the time at 35000. The contracts are quite stable for the following address as the gas remains constant throughout.

2. SCAMS

(A) What is the most lucrative form of scam?



```
# +-------------+---------------------+---------------------+---------
# |     6638809|0x0b6081d38878616...|0x412270b1f0f3884...| 240648550
# |     6638809|0xb43febf2e6c49f3...|0x9eec65e5b998db6...|
class lucrativeScams(MRJob):
    sector = {}

    def steps(self):
        return [MRStep(mapper_init=self.mapper_join_init,
                        mapper=self.mapper_repl_join),
                MRStep(reducer=self.reducer_sum)]

    def mapper_join_init(self):
        with open('scams.json','r') as f:
            file = json.load(f)
        for x in file['result']:
            key = x
            value = file['result'][x]['category']
            self.sector[key] = value

    def mapper_repl_join(self, _, line):
        fields = line.split('\t')
        try:
            if len(fields)==2:
                address = fields[0][1:-1] #to_address
                total_value = int(fields[1]) #value1
                for x in self.sector:
                    if x == address:
                        category = self.sector[x]
                        yield( category, total_value)
                        break
        except:
            pass

    def reducer_sum(self,key,values):
        yield(key,sum(values))


if __name__ == '__main__':
    lucrativeScams.run()
```

JobID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1574975221160_3621/

Input : Using the output of Part B job1 ,i.e., initial aggregation of address and its value.

hadoop :
run2.sh
python $1 -r hadoop --file scams.json --output-dir $3 --no-output
hdfs://andromeda.eecs.qmul.ac.uk/user/hso30$2

```
bash-4.2$ sh run2.sh lucrativeScams.py /outPart_B_job1 partC_scamsAnalysis
```

We perform a replication join on top 10 contracts and the transaction files
Initial steps defined to execute in order:

```
def steps(self):
    return [MRStep(mapper_init=self.mapper_join_init,
                   mapper=self.mapper_repl_join),
            MRStep(mapper=self.mapper_length,
                   reducer=self.reducer_sum)]
```

Mapper:

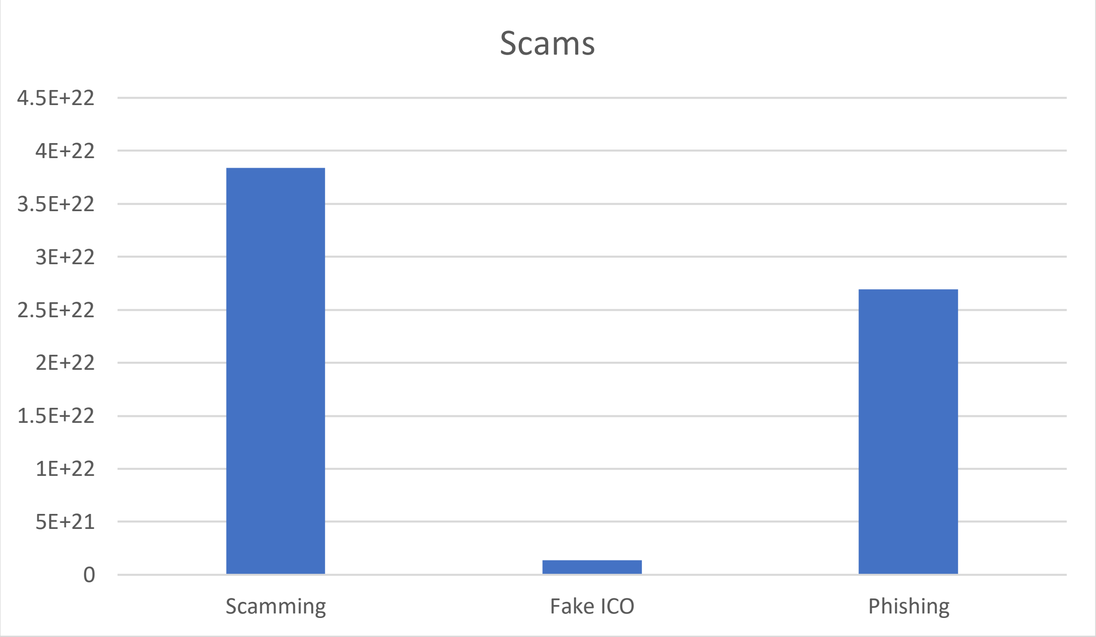
1. mapper_join_init(self)
   a) Stores data from scams.json file where the in sector dictionary, it stores the addresses and the corresponding types of scams.
3. mapper_repl_join(self, _, line)
   a) All the files from part b job 1 are used here.
   b) After verifying whether the given address appear in our dictionary, we yield the category_scam and its aggregate corresponding's value.
   c) yield(category, total_value)

Reducer:

3. Reducer calculates the total value lost as a result of the particular categorical scam.
   a) yield(key,sum(values))

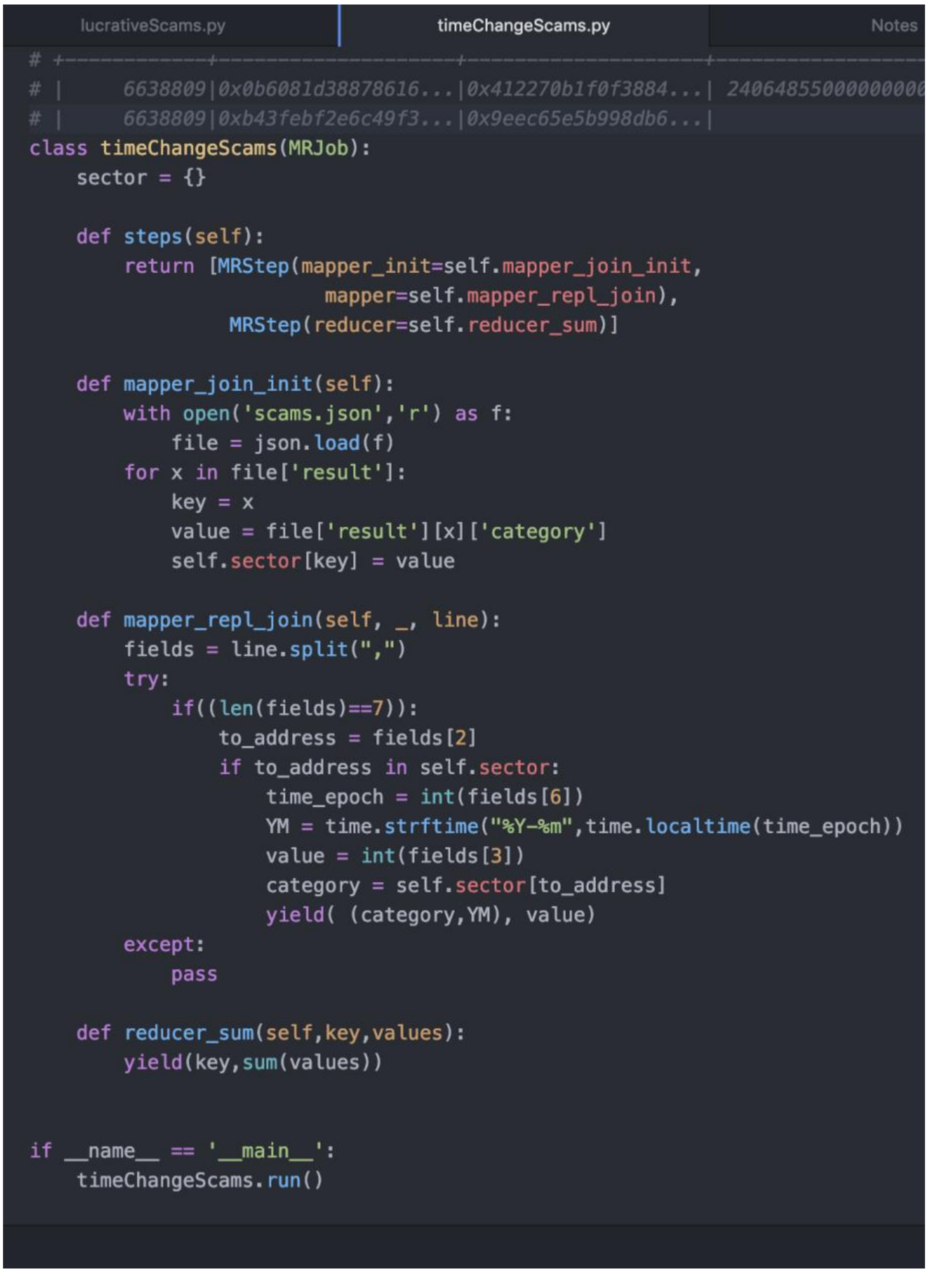
Output:

```
"Scamming" 38407781260421703730344
"Fake ICO"   1356457566889629979678
"Phishing"   26927757396110618476458
```



Observation:
Most value lost through "Scamming" and least value lost through "Fake ICO".

(B) How does this change throughout time



```
# +-------------+--------------------+--------------------+------------------
# |      6638809|0x0b6081d38878616...|0x412270b1f0f3884...| 2406485500000000
# |      6638809|0xb43febf2e6c49f3...|0x9eec65e5b998db6...|
class timeChangeScams(MRJob):
    sector = {}

    def steps(self):
        return [MRStep(mapper_init=self.mapper_join_init,
                         mapper=self.mapper_repl_join),
                MRStep(reducer=self.reducer_sum)]

    def mapper_join_init(self):
        with open('scams.json','r') as f:
            file = json.load(f)
        for x in file['result']:
            key = x
            value = file['result'][x]['category']
            self.sector[key] = value

    def mapper_repl_join(self, _, line):
        fields = line.split(",")
        try:
            if((len(fields)==7)):
                to_address = fields[2]
                if to_address in self.sector:
                    time_epoch = int(fields[6])
                    YM = time.strftime("%Y-%m",time.localtime(time_epoch))
                    value = int(fields[3])
                    category = self.sector[to_address]
                    yield( (category,YM), value)
        except:
            pass

    def reducer_sum(self,key,values):
        yield(key,sum(values))


if __name__ == '__main__':
    timeChangeScams.run()
```

JobID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1574975221160_3909/
 Running job: job_1574975221160_3909

Input : Using the output of Part B job1 ,i.e., initial aggregation of address and its value.

hadoop:
run3.sh
python $1 -r hadoop --file scams.json --output-dir $3 --no-output
hdfs://andromeda.eecs.qmul.ac.uk/$2
bash-4.2$ sh run3.sh timeChangeScams.py /data/ethereum/transactions
partC_scamsTimeChange

We perform a replication join on scam.json file and the transaction files
Initial steps defined to execute in order:

```
def steps(self):
    return [MRStep(mapper_init=self.mapper_join_init,
                   mapper=self.mapper_repl_join),
            MRStep(mapper=self.mapper_length,
                   reducer=self.reducer_sum)]
```

Mapper:

2. mapper_join_init(self)
   a) Stores data from scams.json file where the in sector dictionary, it stores the addresses and the corresponding types of scams.
4. mapper_repl_join(self, _, line)
   a) All the files from part b job 1 are used here.
   b) After verifying whether the given address appear in our dictionary, we calculate the date in readable YM format.
   c) Key = (category,YM) and Value = value from transaction dataset.
   d) yield((category,YM), value)

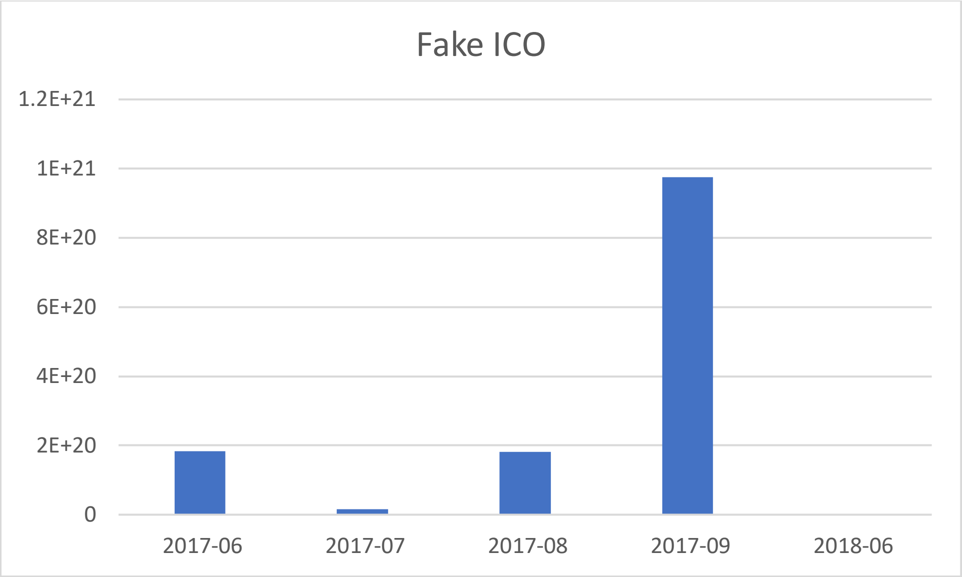
Reducer:

4. Reducer calculates the total value lost as a result of the particular categorical scam.
   a) yield(key,sum(values))
   b) We get a result of total types of scams occurring in particular date and the total value lost as a result of it.

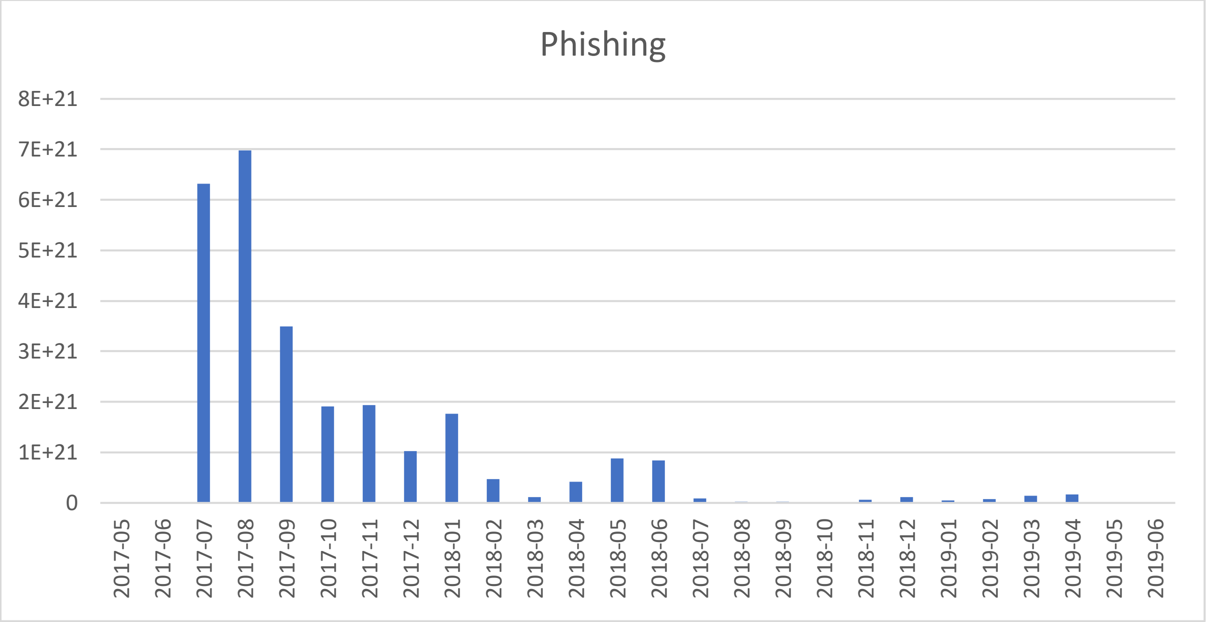
Sample output:

```
["Phishing", "2018-12"]   113564575456080000000
["Phishing", "2019-02"]   75960801308673932726
["Phishing", "2019-04"]   161651973926857053455
["Phishing", "2019-06"]   8087272214976406262
["Scamming", "2017-07"]      1238944359754270060501
["Scamming", "2017-09"]      181698633896218700114
["Scamming", "2017-10"]      1839392288663253070196
["Scamming", "2017-12"]      27509581908918747084
["Scamming", "2018-02"]      498402535182915198977
["Scamming", "2018-04"]      2233443859469805847088
["Scamming", "2018-06"]      1994823614538236819889
["Scamming", "2018-08"]      732512214812598956076
["Scamming", "2018-11"]      180491388841119222190
["Scamming", "2019-01"]      114339300553338156973
["Scamming", "2019-03"]      1450156675337194656091
["Scamming", "2019-05"]      183004992841313501198
["Fake ICO", "2017-07"]   16242199484949186112
["Fake ICO", "2017-09"]   975138363413781359345
```

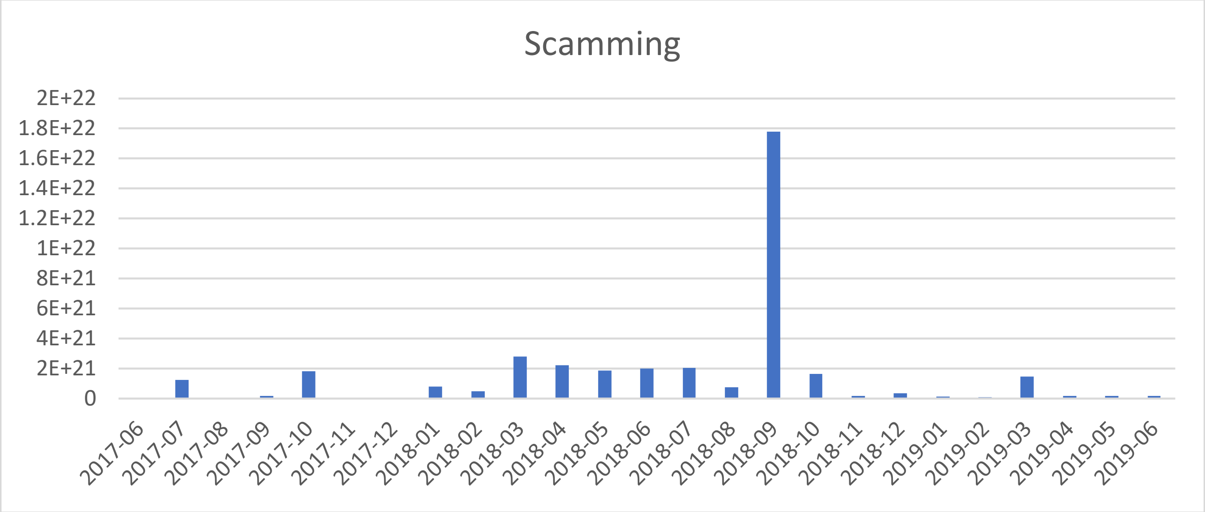
After modifying the output in excel I got the following graph



Observation:
Maximum number of Fake ICO has occurred in 2017-09 costing value of 9.75138E+20.



Observation:
Maximum number of case for Phishing has occurred in 2017-08 costing value of 6.97389E+21



Observation:
Maximum number of case for Phishing has occurred in 2018-09 costing value of 1.78012E+22.