# Ethereum Analysis Coursework

Nathan Ryan Johnson - 170912321

# Part A Time Analysis

**Job ID:**
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_4149/

**Hadoop Command:**
python parta.py -r hadoop hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

**Explanation of MapReduce Program:**
This job is run on the transactions dataset. It is expected that the transactions dataset has 7 fields so this is checked at the start of the mapper. The fields within the dataset are split by commas so this knowledge is defined in the fields variable.

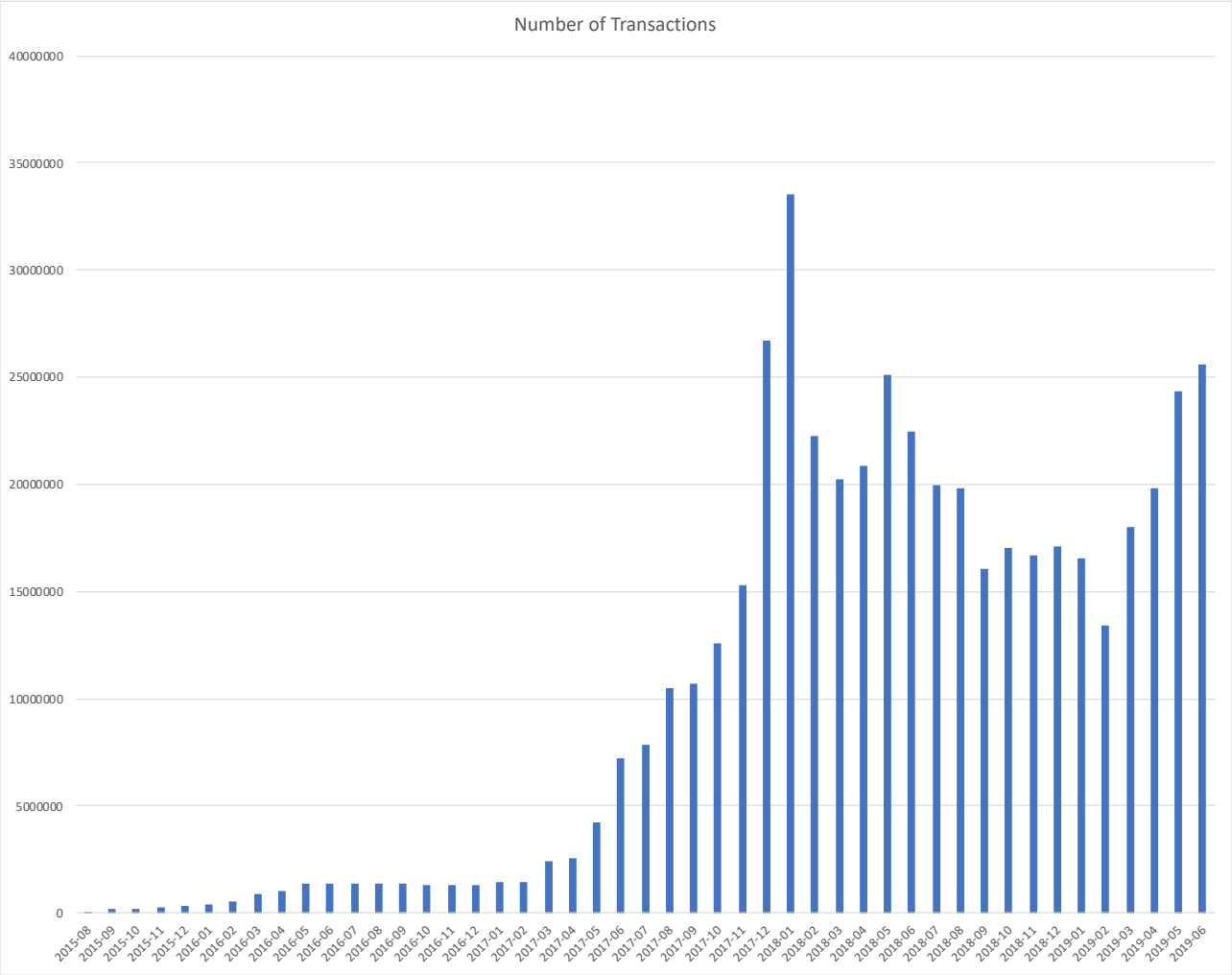
The required field from the dataset is the date. This is yielded with a single transaction.

```
class partA_TimeAnalysis(MRJob):

    def mapper(self, _, line):
        try:
            fields = line.split(",")
            if len(fields) == 7:
                time_epoch = int(fields[6])
                month = time.strftime("%m", time.gmtime(time_epoch))
                year = time.strftime("%Y", time.gmtime(time_epoch))
                year_month = (year, month)
                transaction_count = 1
                yield year_month, transaction_count
        except:
            pass
```

The reducer takes the output from the mapper as input. The reducer yields the month alongside the sum of transaction counts for that month.

```
def reducer(self, month, transaction_count):
    yield month, sum(transaction_count)
```

## Bar Chart:


Number of Transactions
40000000
35000000
30000000
25000000
20000000
15000000
10000000
5000000
0
2015-08
2015-09
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
2016-05
2016-06
2016-07
2016-08
2016-09
2016-10
2016-11
2016-12
2017-01
2017-02
2017-03
2017-04
2017-05
2017-06
2017-07
2017-08
2017-09
2017-10
2017-11
2017-12
2018-01
2018-02
2018-03
2018-04
2018-05
2018-06
2018-07
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
2019-05
2019-06

# Part A Average Transaction Value

**Job ID:**
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_4354/

**Hadoop Command:**
python parta2.py -r hadoop hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

**Explanation:**
This job is run on the transactions dataset. It is expected that the transactions dataset has 7 fields so this is checked at the start of the mapper. The fields within the dataset are split by commas so this knowledge is defined in the fields variable.

The required fields from the dataset is the date and the transaction value. The date is yielded alongside a tuple of the transaction value and the transaction count.

```
class partA2_TimeAnalysis(MRJob):

    def mapper(self, _, line):
        try:
            fields = line.split(",")
            if len(fields) == 7:
                time_epoch = int(fields[6])
                month = time.strftime("%m", time.gmtime(time_epoch))
                year = time.strftime("%Y", time.gmtime(time_epoch))
                year_month = (year, month)
                transaction_count = 1
                transaction_value = int(fields[3])
                yield (year_month, (transaction_value, transaction_count))
        except:
            pass
```

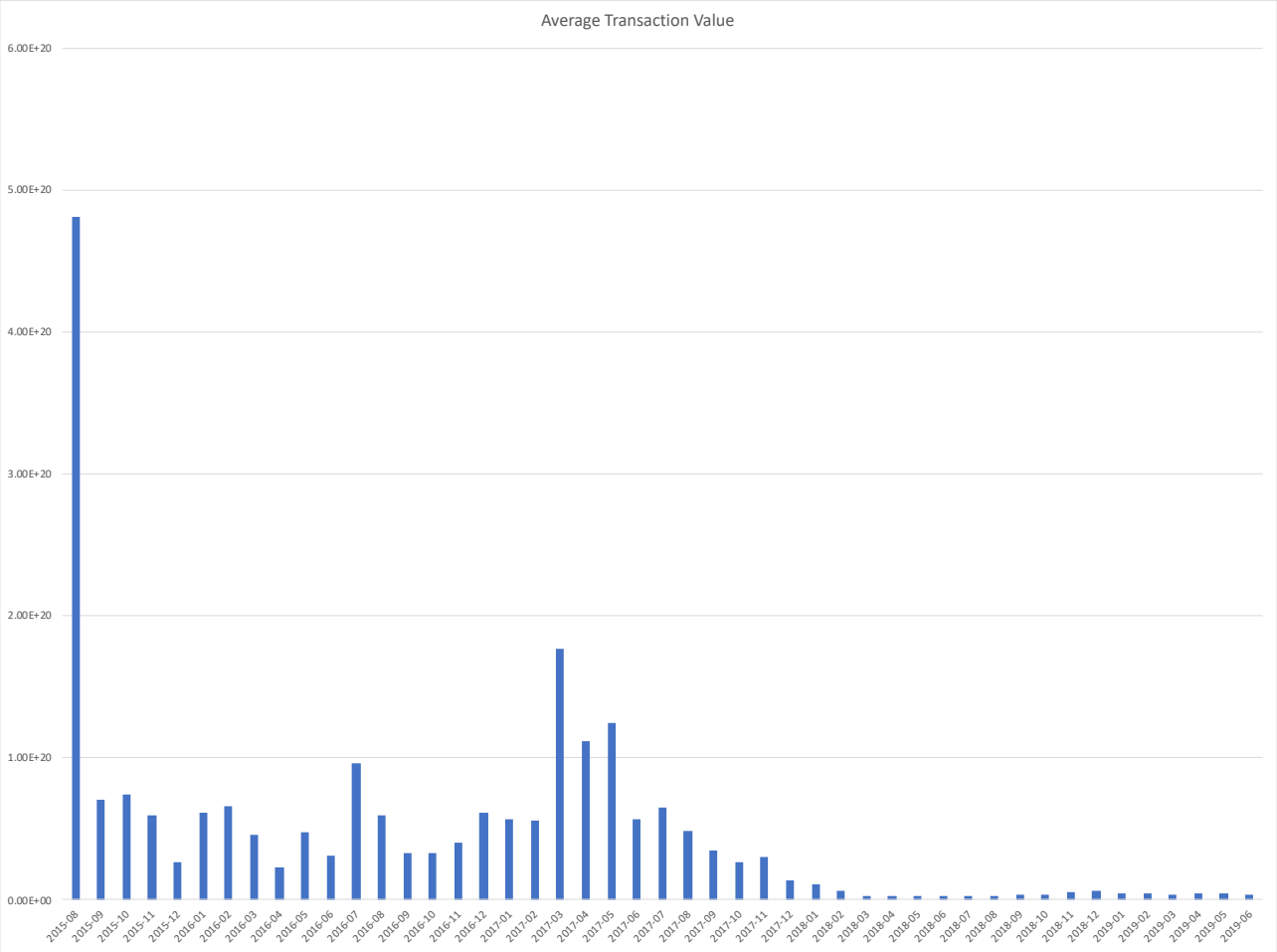
The combiner takes the output from the mapper. The combiner combines each month with its corresponding total value of transactions and total number of transactions. The aim of this combiner is to make the program more efficient by decreasing the workload on the reducer.

```
def combiner(self, month, values):
    count = 0
    total = 0
    for value in values:
        count += value[1]
        total += value[0]
    yield (month, (total,count))
```

The reducer takes the output from the combiner. The reducer yields the current month alongside the average total value of transactions of that month.

```
def reducer(self, month, values):
    count = 0
    total = 0
    for value in values:
        count += value[1]
        total += value[0]
    yield (month, total/count)
```

## Bar Chart:


Average Transaction Value
6.00E+20
5.00E+20
4.00E+20
3.00E+20
2.00E+20
1.00E+20
0.00E+00
2015-08
2015-09
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
2016-05
2016-06
2016-07
2016-08
2016-09
2016-10
2016-11
2016-12
2017-01
2017-02
2017-03
2017-04
2017-05
2017-06
2017-07
2017-08
2017-09
2017-10
2017-11
2017-12
2018-01
2018-02
2018-03
2018-04
2018-05
2018-06
2018-07
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
2019-05
2019-06

# Part B Initial Aggregation

**Job ID:**
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_5897/

**Hadoop Command:**
python partb1.py -r hadoop --output-dir partb1output --no-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

**Explanation:**

This job is run on the transactions dataset. It is expected that the transactions dataset has 7 fields so this is checked at the start of the mapper. The fields within the dataset are split by commas so this knowledge is defined in the fields variable.

```
def mapper(self, _, line):
    try:
        fields = line.split(",")
        if len(fields) == 7:
```

The values of the address and value fields are taken from the transaction datasets.

```
to_address = fields[2]
value = int(fields[3])
```

The values of the address and value fields are yielded.

```
yield(to_address, value)
```

The reducer takes the output of the mapper and then itself yields the address and the sum of values to that address.

```
def reducer(self, address, value):
    yield (address, sum(value))
```

# Part B JOINING TRANSACTIONS/CONTRACTS AND FILTERING

**Job ID:**
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_6354/

**Hadoop Command:**
python partb2.py -r hadoop hdfs://andromeda.eecs.qmul.ac.uk/user/nrj30/partb1textoutput/ --output-dir partb2output --no-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/contracts

**Explanation:**

In this job a repartition join is being performed between the transactions aggregate and Contracts. This will be achieved by performing a join on the to_address from the output of job1 field with the address field of Contracts.

This means that there are two outputs to the mapper function in this job. These two inputs need to be differentiated. The fields in contract are split by commas whereas the fields are split by tabs in job1. Therefore the fields can be separated like so:

```
def mapper(self, _, line):
    try:
        contract_fields = line.split(',')
        job1_fields = line.split('\t')
```

The number of fields in each input are known. The contract dataset has 5 fields and the job1 output has 2 fields. This information is used to in essence create two mappers by using an if elif statement. Each of these 'mappers' we provide a key and a value. In each case the key is the address or to_address field - this is what will be used to join the two datasets. Each case also has a value that is yielded alongside a number representing which input that value came from.

```
        if len(contract_fields)==5:
            key = contract_fields[0]
            value = int(contract_fields[3])
            yield (key, (value,1))


        elif len(job1_fields)==2:
            key = job1_fields[0]
            value = int(job1_fields[1])
            key = key.replace('"', '')
            yield (key,(value,2))
```

The reducer is where the join actually takes place. The reducers takes what the mapper outputted which included the values list. In the reducer this values list is looped through. In this loop it is checked whether the value came from the job1 output or the contracts dataset - this is done by looking at the second value in the value list.  If the value came from the contracts dataset we store this join value in the contracts variable. If the value came from the job1 output we store the value in the total_value field.

```
def reducer(self, address, values):

    contacts = ''
    total_value = ''
    for value in values:
        if value[1] == 1:
            contracts = value[0]
        elif value[1] == 2:
            contracts = contracts
            total_value = value[0]
```

The reducer only yields if the contracts variable has a join value as this job only wants to consider the smart contracts which belong in the contracts

```
    if contracts != '':
        yield (address, total_value)
```

# Part B TOP 10

**Hadoop Command:**
python partb3.py partb2textoutput.txt > partb3textoutput.txt

**Explanation:**

In this mapper the fields need to be defined. The fields in the output partb2textoutput.txt are tab separated and so the fields are split by tabs.

```
def mapper(self, _, line):
    try:
        fields = line.split('\t')
```

Before commencing the rest of the program it is checked that the input contains the expected number of fields. The expected number of fields is 2.

```
        if len(fields) == 2:
```

Address and value are the only two values required.  These two values are taken from the input and stored in variables.

```
            address = fields[0]
            value = int(fields[1])
```

As only address and value are required a None value alongside a tuple containing address and value is yielded.

```
            yield (None, (address, value))
```

A combiner is used to lower the workload of the reducer and thus make the program more efficient. The combiner takes the same input as the reducer will. The combiners first step is to sort the values in descending order of total value.

```
def combiner(self, _, unsorted_values):
    sorted_values = sorted(unsorted_values, reverse=True, key = lambda unsorted_vals:unsorted_vals[1])
```

After the values are sorted the combiner yields the top 10 values from the sorted values. This means the combiner yields the top 10 smart contract addresses alongside a group of values that still need to be reduced.

```
    i=0
    for value in sorted_values:
        yield("Top", value)
        i += 1
        if i >= 10:
            break
```

The reducer takes the output of the combiner. The reducers first step is to sort the values in descending order of total value.

```
def reducer(self, _, unsorted_values):
    sorted_values = sorted(unsorted_values, reverse=True, key = lambda unsorted_vals:unsorted_vals[1])
```

After the values are sorted the reducer yields the top 10 smart contracts with their overall totals.

```
    i=0
    for value in sorted_values:
        yield("{} - {}".format(value[0], value[1]), None)
        i += 1
        if i >= 10:
            break
```

**Output:**

["0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444", 84155100809965865822726776]
["0xfa52274dd61e1643d2205169732f29114bc240b3", 45787484483189352986478805]
["0x7727e5113d1d161373623e5f49fd568b4f543a9e", 45620624001350712557268573]
["0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef", 43170356092262468919298969]
["0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8", 27068921582019542499882877]
["0xbfc39b6f805a9e40e77291aff27aee3c96915bdd", 21104195138093660050000000]
["0xe94b04a0fed112f3664e45adb2b8915693dd5ff3", 15562398956802112254719409]
["0xbb9bc244d798123fde783fcc1c72d3bb8c189413", 11983608729202893846818681]
["0xabbb6bebfa05aa13e908eaa492bd7a8343760477", 11706457177940895521770404]
["0x341e790174e3a4d35b65fdc067b6b5634a61caea", 8379000751917755624057500]

# Part C TOP TEN MOST ACTIVE MINERS

**Job ID:**
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_3127/

**Hadoop Command:**
python partc1.py -r hadoop --output-dir partc1output --no-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/blocks

**Explanation:**
In this mapper the fields need to be defined. The fields in the blocks dataset are comma separated and so the fields are split by commas.

```
def mapper(self, _, line):
    try:
        blocks_fields = line.split(',')
```

The values of the miners and size fields are taken from the transaction datasets.

```
        if len(blocks_fields) == 9:
            miners = blocks_fields[2]
            size = int(blocks_fields[4])
```

The values of the address and value fields are yielded.

```
yield (miners, size)
```

The combiner takes the output from the mapper and decreases the workload for the reducer by combining the output from the mapper.

```
def combiner(self, miners, size):
    yield (miners, sum(size))
```

The reducer takes the output from the combiner and yields each miner with the total size of blocks in bytes that they have mined.

```
def reducer(self, miners, size):
    yield (miners, sum(size))
```

# Part C TOP TEN MOST ACTIVE MINERS

**Hadoop Command:**
python partc2.py partc1textoutput.txt > partc2textoutput.txt

**Explanation:**
In this mapper the fields need to be defined. The fields in the output partc1textoutput.txt are tab separated and so the fields are split by tabs.

```
def mapper(self, _, line):
    try:
        job1_fields=line.split('\t')
```

Before commencing the rest of the program it is checked that the input contains the expected number of fields. The expected number of fields is 2.

```
            if len(fields) == 2:
```

Miners and size are the only two values required.  These two values are taken from the input and stored in variables.

```
                miners = job1_fields[0]
                size = int(job1_fields[1])
                miners = miners.replace('"','')
```

As only miners and size are required a None value alongside a tuple containing miners and size is yielded.

```
yield (None, (miners, size))
```

A combiner is used to lower the workload of the reducer and thus make the program more efficient. The combiner takes the same input as the reducer will. The combiners first step is to sort the values in descending order of size.

```
def combiner(self, _, unsorted_values):
    sorted_values = sorted(unsorted_values, reverse=True, key = lambda unsorted_vals:unsorted_vals[1])
```

After the values are sorted the combiner yields the top 10 values from the sorted values. This means the combiner yields the top 10 miners alongside a group of values that still need to be reduced.

```
    i=0
    for value in sorted_values:
        yield("Top", value)
        i += 1
        if i >= 10:
            break
```

The reducer takes the output of the combiner. The reducers first step is to sort the values in descending order of total value.

```
def reducer(self, _, unsorted_values):
    sorted_values = sorted(unsorted_values, reverse=True, key = lambda unsorted_vals:unsorted_vals[1])
```

After the values are sorted the reducer yields the most active miners with the size of the blocks in bytes that they mined.

```
    i = 0
    for values in sorted_values:
        yield("{} - {}".format(values[0], values[1]), None)
        i += 1
        if i >= 10:
            break
```

**Output:**

"0xea674fdde714fd979de3edf0f56aa9716b898ec8 - 23989401188"
"0x829bd824b016326a401d083b33d092293333a830 - 15010222714"
"0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c - 13978859941"
"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5 - 10998145387"
"0xb2930b35844a230f00e51431acae96fe543a0347 - 7842595276"
"0x2a65aca4d5fc5b5c859090a6c34d164135398226 - 3628875680"
"0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01 - 1221833144"
"0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb - 1152472379"
"0x1e9939daaad6924ad004c2560e90804164900341 - 1080301927"
"0x61c808d82a3ac53231750dadc13c777b59310bd9 - 692942577"

# Part D GAS GUZZLERS

**Job ID:**
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_6870/

**Hadoop Command:**
python partdGasGuzzlers.py -r hadoop --output-dir partdGasGuzzlerOutput --no-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

**Explanation:**
In this mapper the fields need to be defined. The fields in the blocks dataset are comma separated and so the fields are split by commas.

```
class PartDGazGuzzlersJob1(MRJob):

    def mapper(self, _, line):
        fields = line.split(",")
```

The date and the gas price are the two pieces of data that are needed. Both of these pieces of data are extracted from the transactions dataset. They are both yielded at the end of the mapper.

```
        try:
            if((len(fields)==7)):
                date = time.localtime(int(fields[6]))
                year = date[0]
                month = date[1]
                gas_price = int(fields[5])
                year_month = (year, month)
                yield(year_month,gas_price)
```

The reducer takes what is outputted from the mapper and yield the date (year and month) alongside the average (mean) gas price for that date.

```
def reducer(self, year_month, gas_price):
    yield(year_month,statistics.mean(gas_price))
```

# Part D GAS GUZZLERS

**Job IDs:**
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_8473/
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_8498/

**Hadoop Command:**
python partdGasGuzzlers2.py -r hadoop --file partb3textoutput.txt --output-dir partdGasGuzzlerOutput2 --no-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

**Explanation:**
In this part the steps that the program will follow are defined.

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

This program takes the top 10 smart contracts from part b as input. This file is read in to the first mapper. Here there is a for loop that stere the data from the top 20 smart contracts in the sector mapper as sector[address]. Here address is the key and the value (fields[1]) is the total aggregate obtained in the top 10 smart contracts file.

```
def mapper_join_init(self):
    with open("partb3textoutput.txt") as f:
        for line in f:
            address_value = line.split("\t")
            fields = address_value[0].split(",")
            address = fields[0][2:-1]
            self.sector[address] = fields[1]
```

Here the transactions dataset is joined with the top 10 smart contacts. This join only occurs if the to_address is equal to the address of the top 10 contracts transactions. At the end of the mapper the address and date is yielded alongside the transaction value.

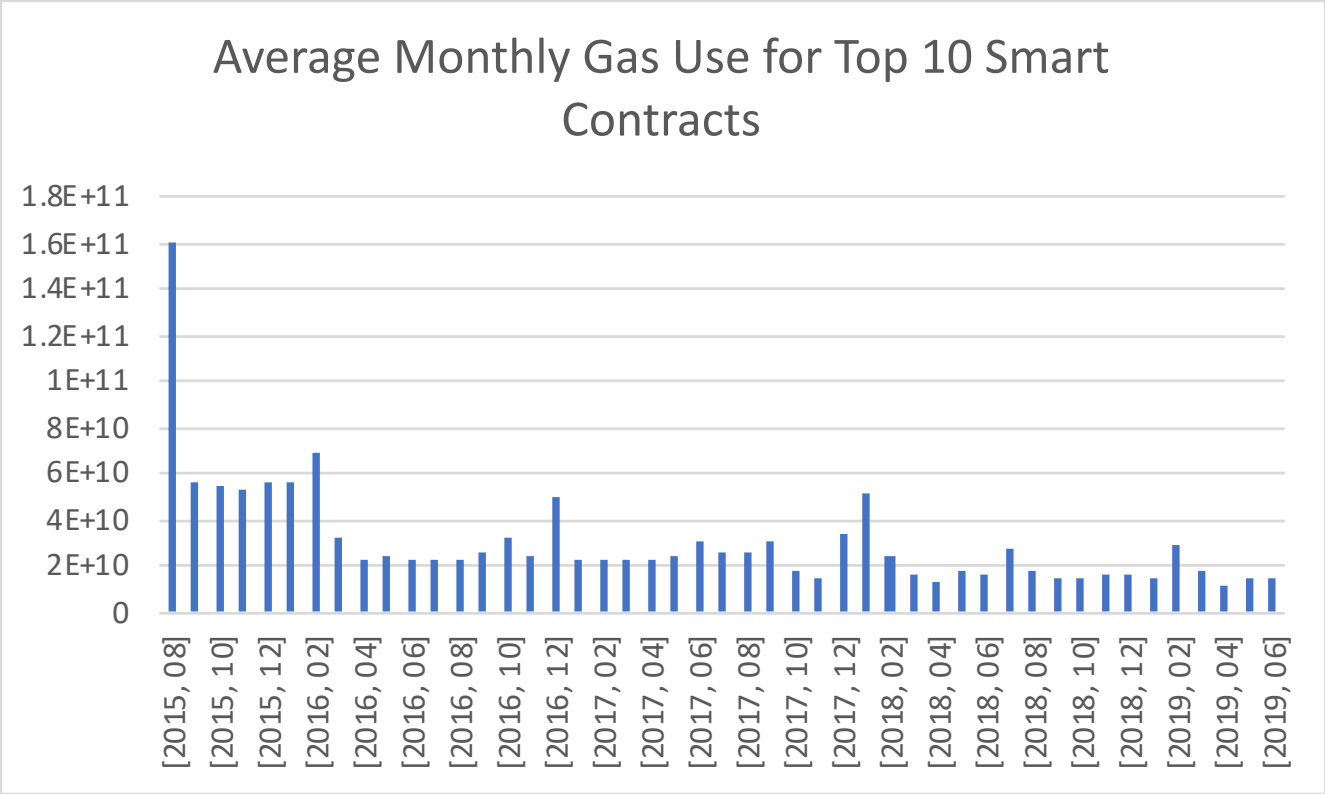
```
def mapper_repl_join(self, _, line):

    fields = line.split(",")
    try:
        if((len(fields)==7)):
            to_address = fields[2]
            if to_address in self.sector:
                transaction_value = int(fields[3])
                time_epoch = int(fields[6])
                year = time.strftime("%Y",time.localtime(time_epoch))
                month = time.strftime("%m",time.localtime(time_epoch))
                year_month = (year, month)
                address_date = (to_address,year_month)
                yield (address_date, transaction_value)
    except:
        pass
```

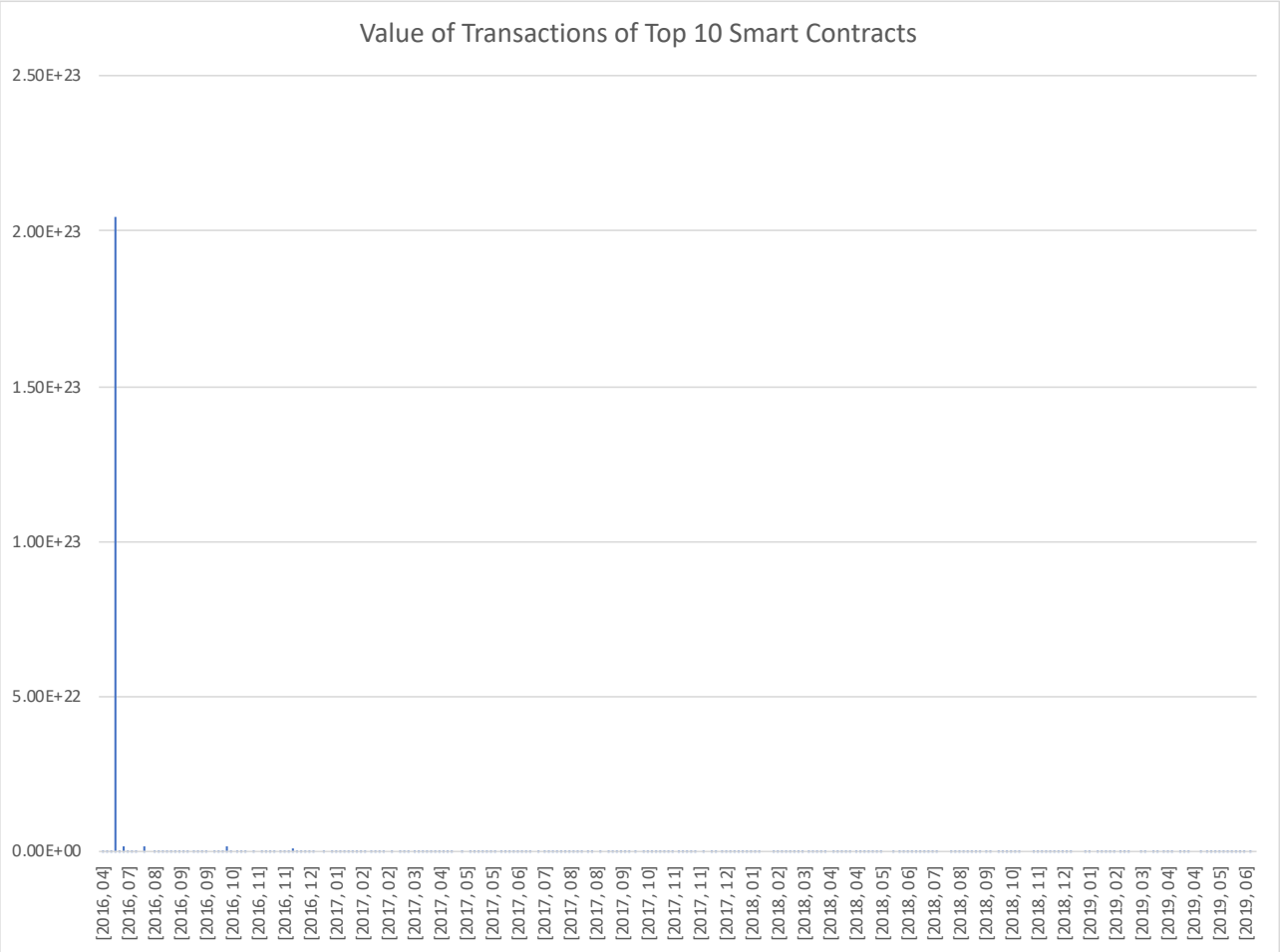
The reducer yields the date alongside the average (mean) gas price for that date.

```
def reducer_sum(self,address_date,transaction_value):
    yield(address_date,statistics.mean(transaction_value))
```

**Output:**

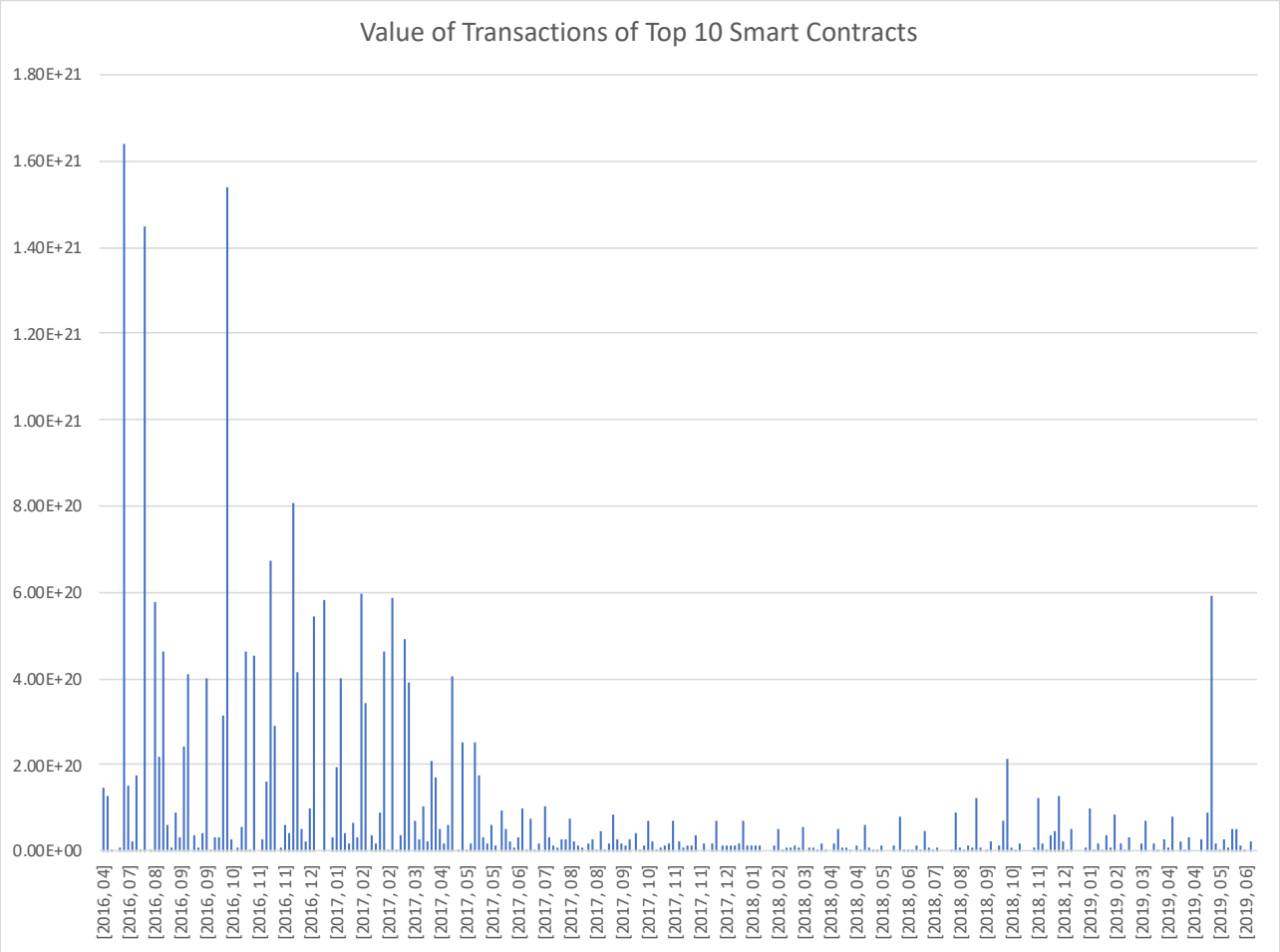
This is the Average Monthly Gas Use for Top 10 Smart Contracts



**Observation:**
The average monthly gas use for the top 10 smart contacts, while it has spikes, is on a general down trend.

In the about graph one datapoint is so large that it is hard to see the trend. To show the trend in the graph below this datapoint has been removed.




Once this data point is removed it can be seen that there is a correlation between the average monthly gas use for the top 10 smart contacts and the value of transactions of top 10 smart contracts. They both show downward trends.

# Part D POPULAR SCAMS

**Job IDs:**
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_7518/
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_7541/

**Hadoop Command:**
python partdScams1.py -r hadoop --file scams.json --output-dir partdScamsOutput1 --no-output partb1textoutput.txt

**Explanation:**
In this part the steps that the program will follow are defined.

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

This stores data from the scams.json file in a dictionary. It stores address as a key and type of scam as a value.

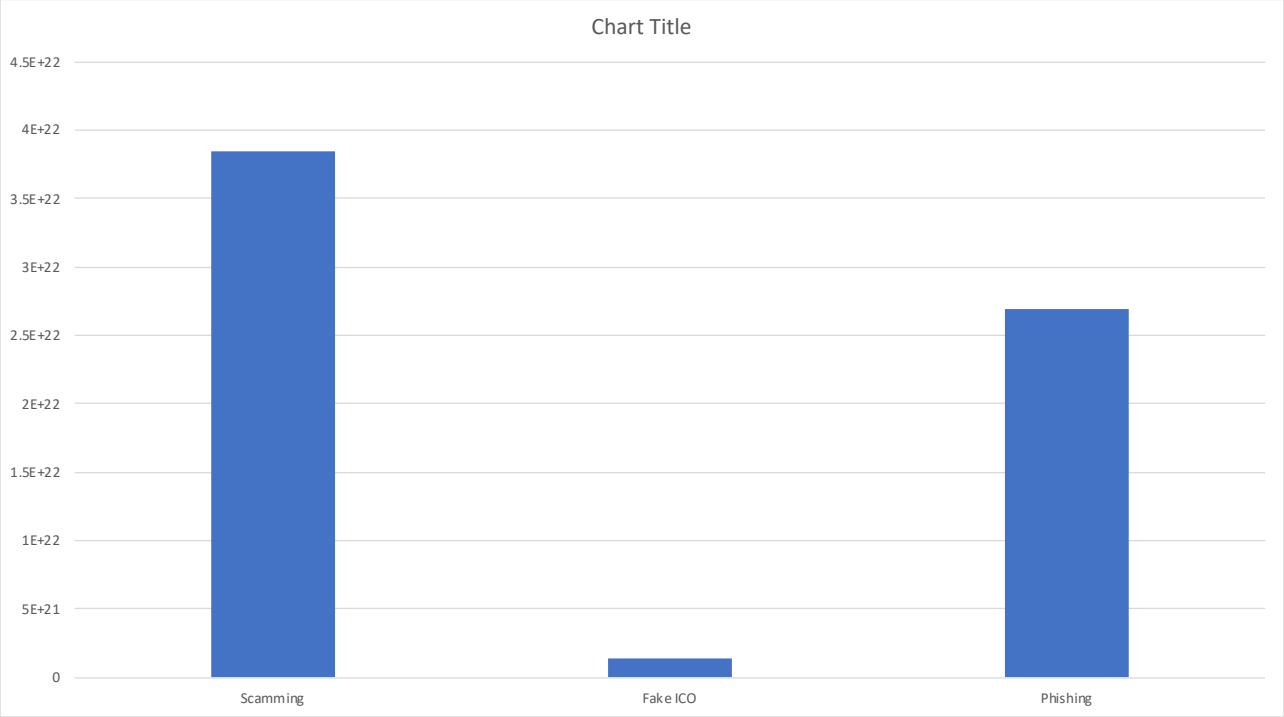
```
def mapper_join_init(self):
    with open('scams.json','r') as f:
        file = json.load(f)
    for key in file['result']:
        value = file['result'][key]['category']
        self.sector[key] = value
```

Here the output of b part 1 is used. If a given address is in the dictionary created in the mapper_init then the category of scam is yielded alongside its total_value.
The reducer sums the total value lost for each category of scam.

```
def reducer_sum(self,category,total_value_lost):
    yield(category,sum(total_value_lost))
```

**Output:**

| | |
|---|---|
| Scamming | 3.84078E+22 |
| Fake ICO | 1.35646E+21 |
| Phishing | 2.69278E+22 |




From the data it appears that Scamming was the most lucrative form of scam in a given month.

# Part D POPULAR SCAMS - HOW IT CHANGES THROUGHOUT TIME

**Job IDs:**
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_7602/
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_7608/

**Hadoop Command:**
python partdScams2.py -r hadoop --file scams.json --output-dir partdScamsOutput2 --no-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

**Explanation:**
In this part the steps that the program will follow are defined.

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

This stores data from the scams.json file in a dictionary. It stores address as a key and type of scam as a value.

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

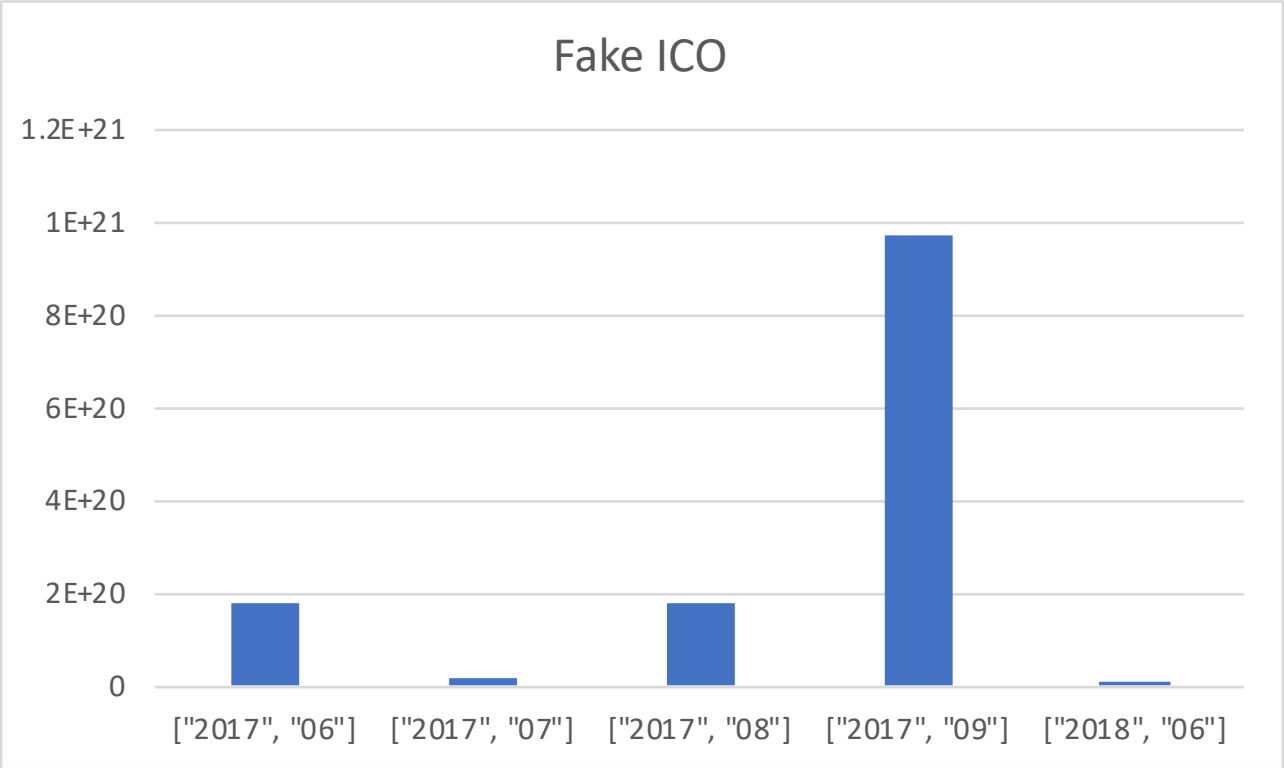
Here the output of b part 1 is used. If a given address is in the dictionary created in the mapper_init then the category of scam is yielded in a tuple, with the date of the transaction, alongside its total_value.

```
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])
                year = time.strftime("%Y",time.localtime(time_epoch))
                month = time.strftime("%m",time.localtime(time_epoch))
                year_month = (year, month)
                value = int(fields[3])
                category = self.sector[to_address]
                yield( (category,year_month), value)
    except:
        pass
```

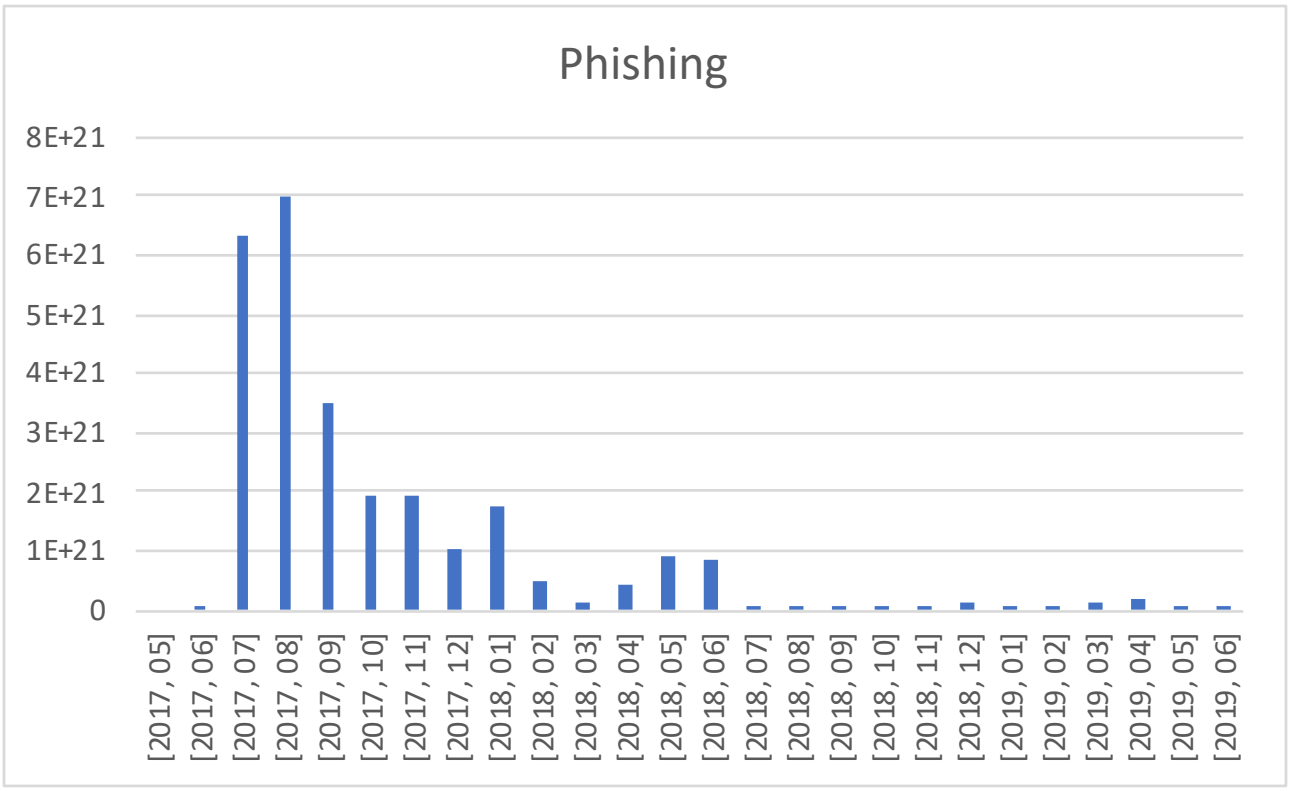
The reducer sums the total value lost for a given type of scam at a given date.

```
def reducer_sum(self,category_year_month,total_value_lost):
    yield(category_year_month,sum(total_value_lost))
```

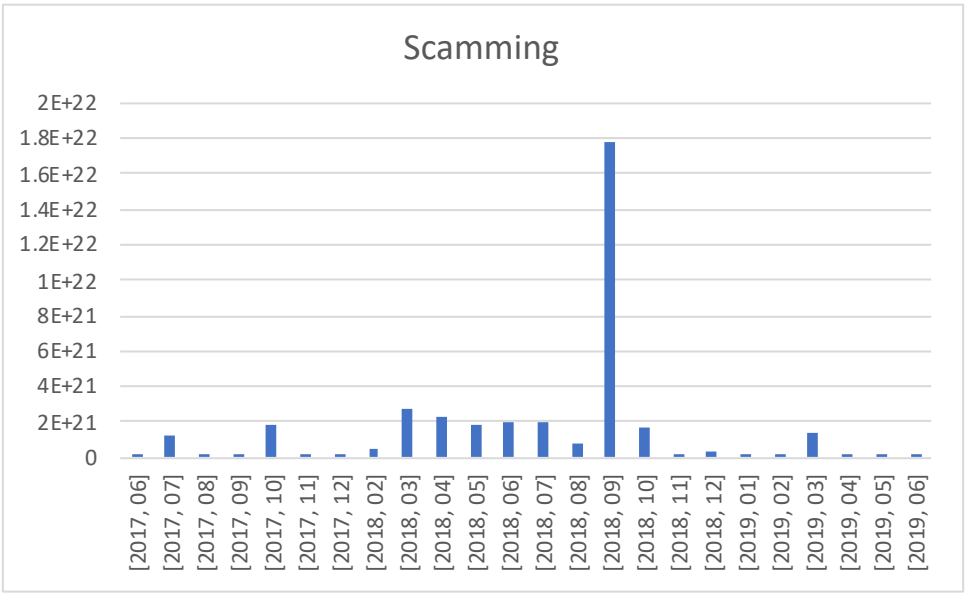
**Explanation:**




Observation: Maximum number of Fake ICO scams occurred in September 2017 losing a total value of 9.75E+20




Observation: Maximum number of Phishing scams occurred in September 2017 losing a total value of 6.97E+21.




Observation: Maximum number of Scamming cases occurred in September 2017 losing a total value of 2.79E+21.

**Observation:**

Jack Morse (2020) says that 'almost 80 percent of 2017's ICO were "identified scams."' This correlates with the data that is seen in the Fake ICO bar chart as there is a huge peak in 2017.

# References

Jack Morse. 2020. Almost 80 percent of 2017's ICOs were scams. [ONLINE] Available at: https://mashable.com/article/majority-ico-scams/?europe=true. [Accessed 13 December 2020].