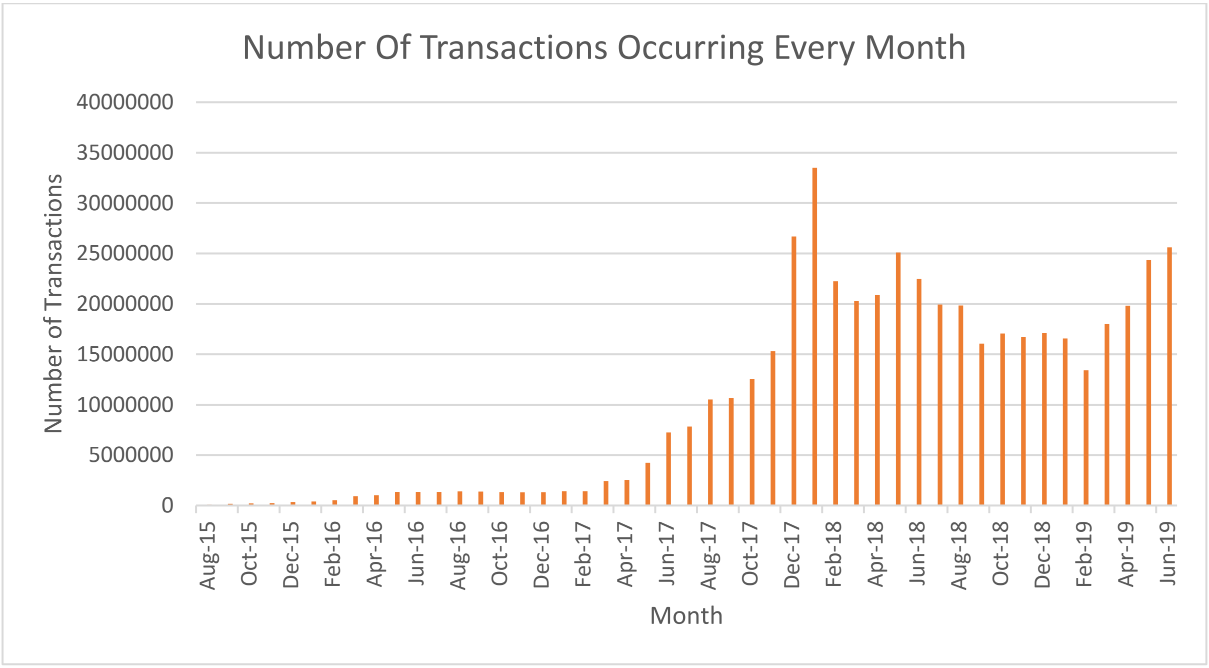
Big Data Coursework

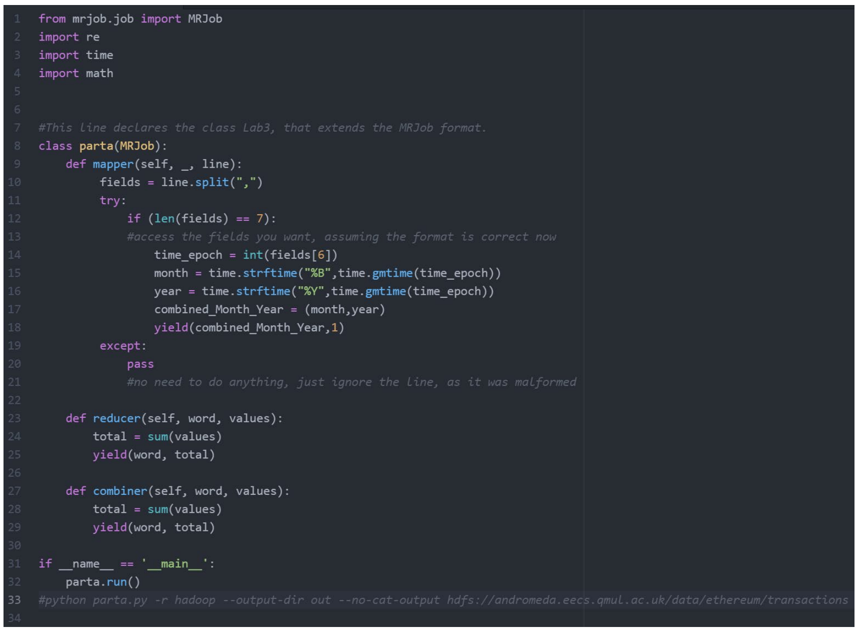
SID: 190165987

Part A(i): Create a bar plot showing the number of transactions occurring every month between the start and end of the dataset.



The following Bar Plot shows the number of transactions occurring every month between start and end of dataset. The data reveals that the monthly trend in the number of transactions is low at first, then gradually increases, peaking between November and December 2017 before declining and climbing again between April and June 2019.

The following graph and conclusion were possible due to this code:



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


#This line declares the class Lab3, that extends the MRJob format.
class parta(MRJob):
    def mapper(self, _, line):
         fields = line.split(",")
         try:
             if (len(fields) == 7):
             #access the fields you want, assuming the format is correct now
                 time_epoch = int(fields[6])
                 month = time.strftime("%B",time.gmtime(time_epoch))
                 year = time.strftime("%Y",time.gmtime(time_epoch))
                 combined_Month_Year = (month,year)
                 yield(combined_Month_Year,1)
         except:
             pass
             #no need to do anything, just ignore the line, as it was malformed

    def reducer(self, word, values):
        total = sum(values)
        yield(word, total)

    def combiner(self, word, values):
        total = sum(values)
        yield(word, total)

if __name__ == '__main__':
    parta.run()
#python parta.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions
```

- I start by importing the right libraries such as time, MRjob, math and regular expression. I then define my class with 3 methods a mapper, reducer, and a combiner.

The Mapper:

- Each record in the dataset is split by a comma, I put the code in try and catch statement just in case the lines received from the splits were malformed.
- Then I check the format of the data is correct by checking the number of fields matches the number of fields in the schema. For the transaction table there are 8 fields in the schema and as computers count from 0, I check if the number of fields is 7. From schema I know that in field position 7 is the time epoch, so I get the epoch time from field position 6 in the program.
- Then I extract the month and year of the transaction and put them in their respective variables. Then I combine the month and year in a tuple, before yielding the tuple and the number 1, where the number 1 is representing one transaction.

The Reducer:

- Each number 1 in the values parameter is then added together using the built in python “sum()” function and stored in the total variable.
- Then the parameter word and total are yielded. Word contains the tuple “month and year” and total contains the count of all the transactions for the month and year in the tuple.

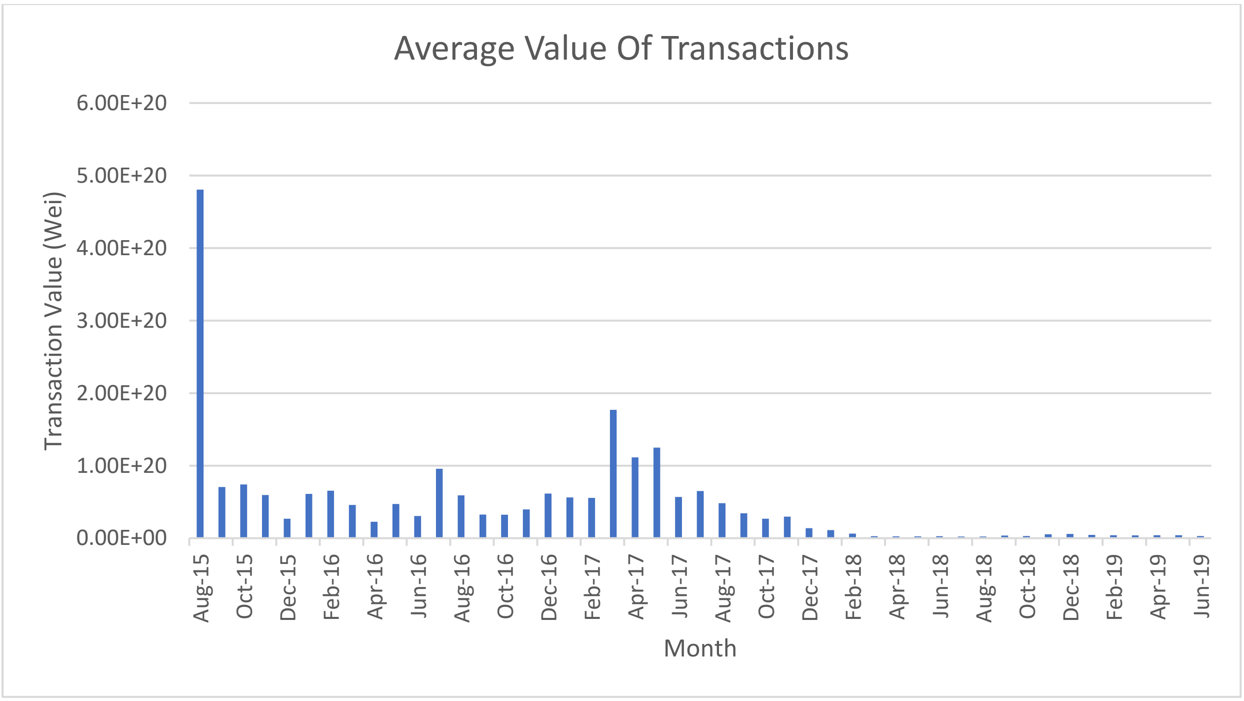
The Combiner:

- Has the same code as the reducer, because in this case we are only using the combiner to get key value pairs from the mapper so (tuple containing month + year, transactions for that month and year), then passing this information to the reducer.
- The Combiner will operate like a Reducer, but only on the subset of the Key/Values output from each Mapper.

Command to run: `python parta.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions`

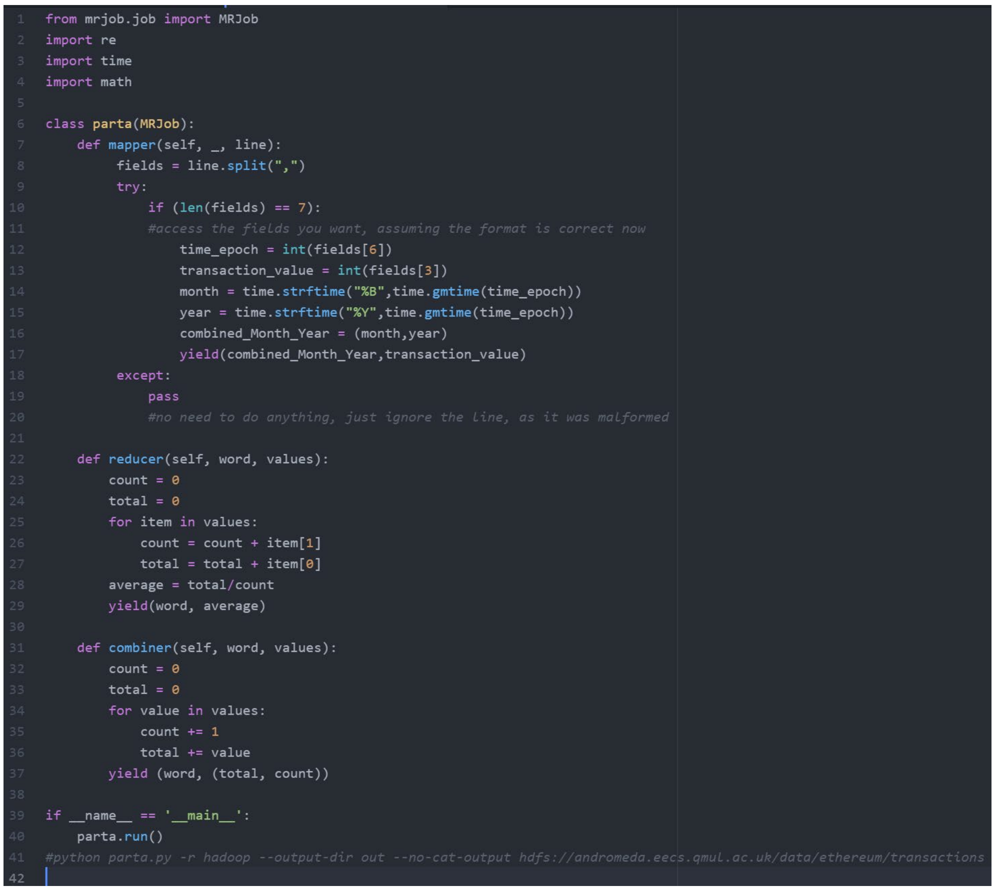
File name: PartA/parta.py

Part A(ii): Create a bar plot showing the average value of transaction in each month between the start and end of the dataset.



The following Bar Plot shows the average value of transactions occurring every month between start and end of dataset. The data reveals that the average transaction value had peaked in August 2015, at the beginning of the dataset, then declining sharply for the next two years until March 2017, when it increased marginally before falling again.

The following graph and conclusion were possible due to this code:



```
from mrjob.job import MRJob
import re
import time
import math

class parta(MRJob):
    def mapper(self, _, line):
         fields = line.split(",")
         try:
             if (len(fields) == 7):
             #access the fields you want, assuming the format is correct now
                 time_epoch = int(fields[6])
                 transaction_value = int(fields[3])
                 month = time.strftime("%B",time.gmtime(time_epoch))
                 year = time.strftime("%Y",time.gmtime(time_epoch))
                 combined_Month_Year = (month,year)
                 yield(combined_Month_Year,transaction_value)
         except:
             pass
             #no need to do anything, just ignore the line, as it was malformed

    def reducer(self, word, values):
        count = 0
        total = 0
        for item in values:
            count = count + item[1]
            total = total + item[0]
        average = total/count
        yield(word, average)

    def combiner(self, word, values):
        count = 0
        total = 0
        for value in values:
            count += 1
            total += value
        yield (word, (total, count))

if __name__ == '__main__':
    parta.run()
#python parta.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions
```

The Mapper:

- Each record in the dataset is separated by a comma.
- In case the lines received from the splits were faulty, I placed the code in try and catch statements.
- Then I double-check that the data is in the correct format by comparing the number of fields to the number of fields in the schema. I check that the number of fields is 7.
- From schema I know that in field position 7 is the time epoch and position 4 is the transaction value, so I get the epoch time from field position 6 in the program and transaction value from field position 3 and store them in their respective variables.
- Then I extract the transaction's month and year and store them in their appropriate variables, making a tuple out of the month and year.
- Then I yield the tuple and the transaction value, with the transaction value denoting a single transaction that occurred during the month and year indicated in the tuple.

The Combiner:

- Two variables are declared a count and a total.
- I iterate through each transaction value in the parameter values for that specific month and year.
- Each loop iteration I increase the value of the count by one and add the transaction value to the current total.
- Then the parameter word and the average are yielded. Word contains the tuple "month and year", and average contains the average value of transactions for the month and year in the tuple.
- Then the parameter word and the (total,count) is yielded for use in the reducer. So, essentially, I created a subset of key and value pairs for the reducer to use.

The Reducer:

- A count and a total have been declared as variables.
- I then iterate through each transaction value in the parameter values for that specific month and year using a for loop, incrementing the count by one with each loop iteration and add the transaction value to the current total.
- Then I compute the average by putting the total over the count to receive the average value of transaction for that one month.
- Then the parameter word and the average are yielded. Word contains the tuple "month and year", and average contains the average value of transactions for the month and year in the tuple.

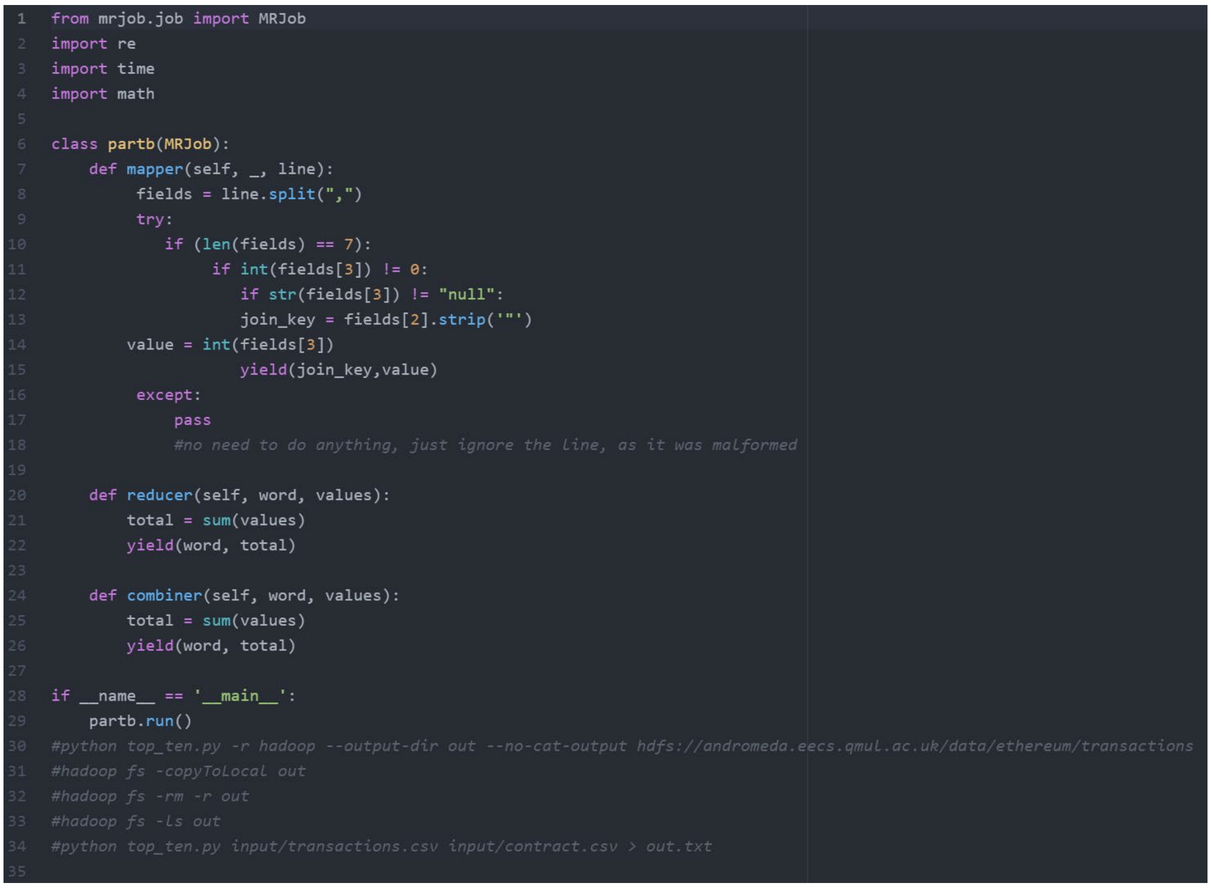
Command to run: `python parta.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions`

File name: PartA/ parta_average.py

Part B is split into 3 jobs to get the answer.

Part B(Job1):

In Job1 I aggregate all the transactions to see how much each address within the user space has been involved, very similar to what was done in wordcount.py in lab1. This was possible by the following code:



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

class partb(MRJob):
    def mapper(self, _, line):
         fields = line.split(",")
         try:
            if (len(fields) == 7):
                 if int(fields[3]) != 0:
                    if str(fields[3]) != "null":
                    join_key = fields[2].strip('"')
        value = int(fields[3])
                    yield(join_key,value)
         except:
             pass
             #no need to do anything, just ignore the line, as it was malformed

    def reducer(self, word, values):
        total = sum(values)
        yield(word, total)

    def combiner(self, word, values):
        total = sum(values)
        yield(word, total)

if __name__ == '__main__':
    partb.run()
#python top_ten.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions
#hadoop fs -copyToLocal out
#hadoop fs -rm -r out
#hadoop fs -ls out
#python top_ten.py input/transactions.csv input/contract.csv > out.txt
```

The Mapper:

- The schema for the transaction table has 8 elements, I check that the number of fields is 7, I also check the fields[3] which is the value is not null or 0.
- Then I extract the transaction destination addresses and strip quotation marks from the addresses in field position 2 as I will be using this data to join on other data in another job.
- Then I extract the value of transaction in field position 3 and yield both the join key and the value.

The Reducer:

- Each transaction value in the values parameter is then added together using the built in python “sum()” function and stored in the total variable.
- Then the parameter word and total are yielded. Word contains the destination addresses of transaction, so the total is the total value of the transactions to that specific destination address.

The Combiner:

- I used the same code as the reducer, because in this case we are only using the combiner to get key value pairs from the mapper so the destination address, then passing this information to the reducer.
- The Combiner will operate like a Reducer, but only on the subset of the Key/Values output from each Mapper.

Command to run: python top_ten.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

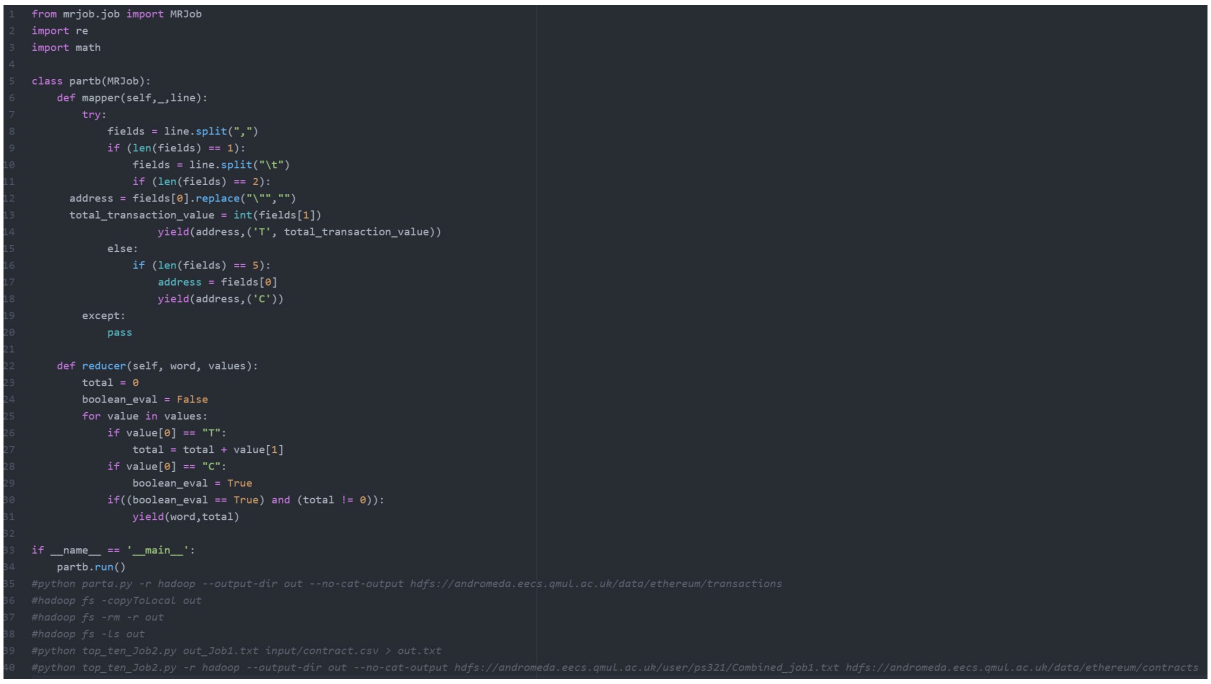
File name: PartB/top_ten.py

Part B(Job2):

Before starting this job, I used the Hadoop get merge command to get a merged output file from job1 then I used copy to Hadoop filesystem command to copy the merged file back into Hadoop file system ready to perform a join on it with the next job.

In Job2 I perform a repartition join between aggregates calculated in the destination field in contracts while also filtering out any user addresses so only smart addresses are left.

This was possible by the following code:



```python
from mrjob.job import MRJob
import re
import math

class partb(MRJob):
    def mapper(self,_,line):
        try:
            fields = line.split(",")
            if (len(fields) == 1):
                fields = line.split("\t")
                if (len(fields) == 2):
      address = fields[0].replace("\"","")
      total_transaction_value = int(fields[1])
                    yield(address,('T', total_transaction_value))
            else:
                if (len(fields) == 5):
                    address = fields[0]
                    yield(address,('C'))
        except:
            pass

    def reducer(self, word, values):
        total = 0
        boolean_eval = False
        for value in values:
            if value[0] == "T":
                total = total + value[1]
            if value[0] == "C":
                boolean_eval = True
            if((boolean_eval == True) and (total != 0)):
                yield(word,total)

if __name__ == '__main__':
    partb.run()
#python parta.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions
#hadoop fs -copyToLocal out
#hadoop fs -rm -r out
#hadoop fs -ls out
#python top_ten_Job2.py out_Job1.txt input/contract.csv > out.txt
#python top_ten_Job2.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/user/ps321/Combined_job1.txt hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/contracts
```

The Mapper:

- I check if the file is the out.txt file from the previous job by checking if the length of fields is 1. Then if it's the case I split the record of the dataset by "/t" as the out.txt file has whitespaces between the values.
- Now that I have split the data, I check if the dataset is the out.txt by checking if the length is 2 and if it is I yield the address with removed backslashes with replaced

commas and the total transaction value and the value T as this address is a transaction.
- Now if the dataset is not the out.txt then I check if the dataset is of length 5 to check if it's the contracts dataset, if it is then I extract the address from field position 0 in contracts and yield it with the letter C denoting it's a contract.

This job has no combiner as it is using a repartition join.

The reducer:

- Firstly, I declare a total variable to 0 and a Boolean flag variable to false.
- Then I iterate for each value in the values parameter to check if the record is of a transaction by checking if position 0 of value is a T. If it is, I add the transaction value to the total.
- Then I check if the record is of a contract by checking if position 0 of value is a C. If it is I change the Boolean flag to True, so only transactions that are smart contracts are yielded.
- Finally, I do another if statement if Boolean flag is true and the total of transaction is not 0, I yield the word containing the address and the total value of all the transactions sent to that address.

Command to run after environment is set up: python top_ten_Job2.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/user/ps321/Combined_job1.txt hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/contracts

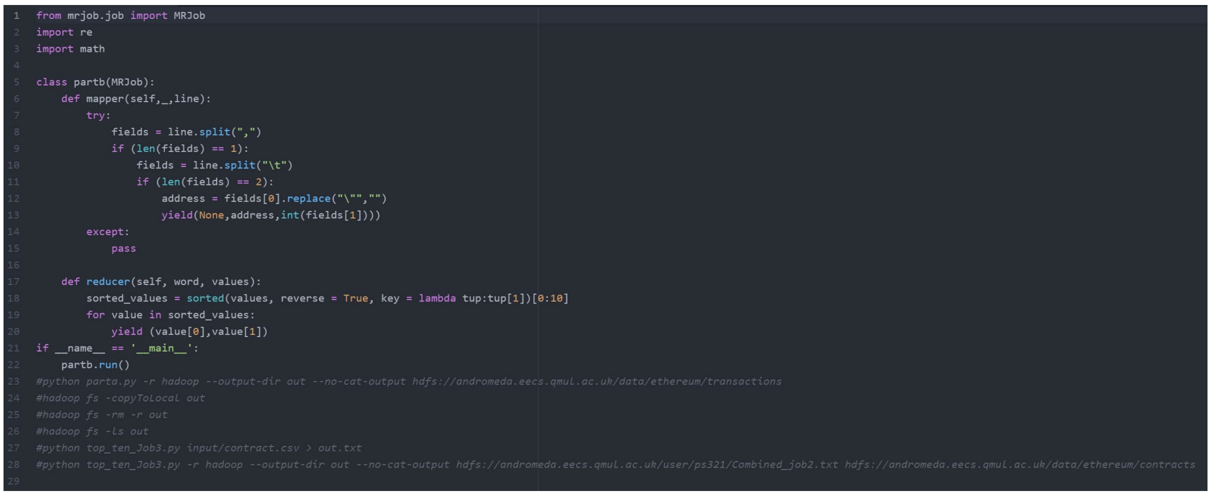
File name: PartB/top_ten_Job2.py

Part B(Job3):

Before starting this job, I used the Hadoop get merge command to get a merged output file from job2 then I used copy to Hadoop filesystem command to copy the merged file back into Hadoop file system ready to perform a top 10 job on it.

In job3, I take the recently joined output of the dataset and find the top 10 most popular services.

This was possible by the following code:



```python
from mrjob.job import MRJob
import re
import math

class partb(MRJob):
    def mapper(self,_,line):
        try:
            fields = line.split(",")
            if (len(fields) == 1):
                fields = line.split("\t")
                if (len(fields) == 2):
                    address = fields[0].replace("\"","")
                    yield(None,address,int(fields[1]))
        except:
            pass

    def reducer(self, word, values):
        sorted_values = sorted(values, reverse = True, key = lambda tup:tup[1])[0:10]
        for value in sorted_values:
            yield (value[0],value[1])
if __name__ == '__main__':
    partb.run()
#python parta.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions
#hadoop fs -copyToLocal out
#hadoop fs -rm -r out
#hadoop fs -ls out
#python top_ten_Job3.py input/contract.csv > out.txt
#python top_ten_Job3.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/user/ps321/Combined_job2.txt hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/contracts

```

The Mapper:

- I check if the file is the out.txt file from the previous job by checking if the length of fields is 1. Then if it's the case I split the record of the dataset by "/t" as the out.txt file has whitespaces between the values.
- As I have now split the dataset, I check if the dataset is out.txt by checking if the length is 2 and if it is I yield the address with removed backslashes with replaced commas and the total transaction value and a None type, none type is because of the sorting taking place in the reducer.

The reducer:

- I use the built in python sorted() function, where it takes the values parameter as the data to sort. I have reverse = True to ensure the data is sorted in reverse. I have a key equal to the lambda function which would return the position[1] of the values as we want to sort by transaction value. I also include a [0:10] so the command is only run 10 times.
- Finally, I iterate over sorted values yielding sorted_values[0] and sorted_values[1] which is the address and the total transaction value of the top 10 smart contracts, which gives us the top 10 most popular services.

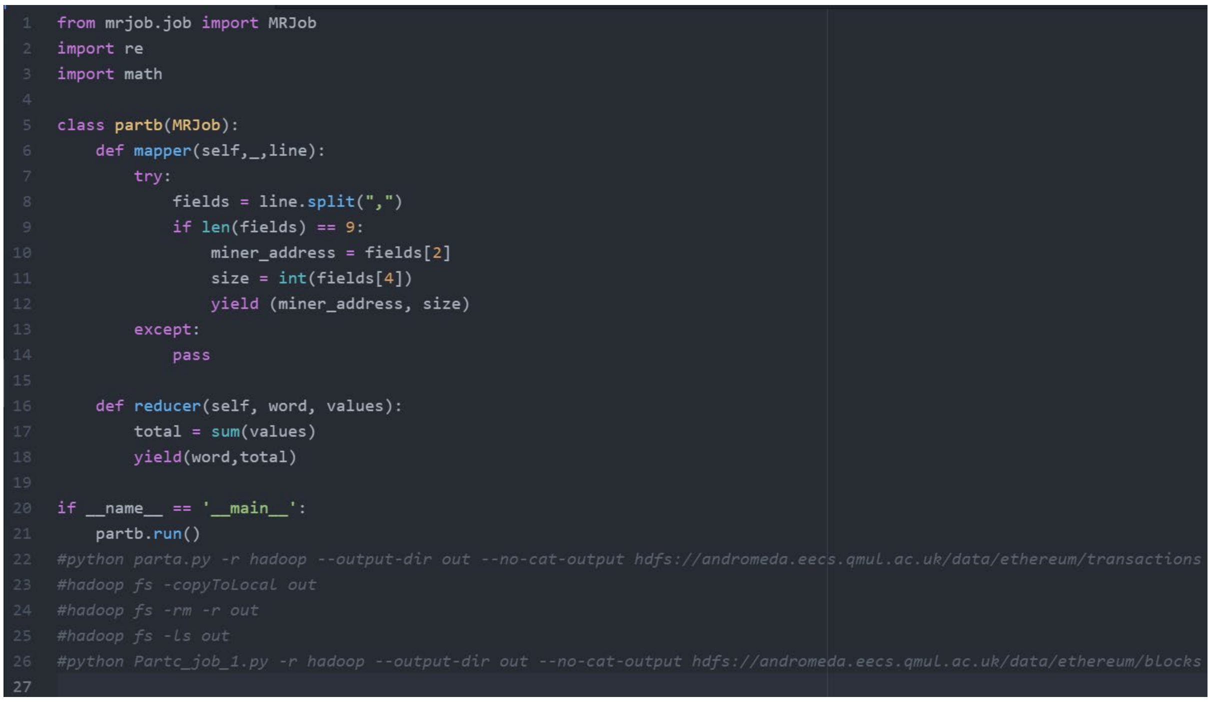
Command to run after environment is set up: python top_ten_Job3.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/user/ps321/Combined_job2.txt hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/contracts

File name: PartB/top_ten_Job3.py

Output file: PartB/top_ten_out.txt

Part C(Job1):

Firstly, I aggregate the blocks to see how much each miner has been involved. The following code explains how:



```python
from mrjob.job import MRJob
import re
import math

class partb(MRJob):
    def mapper(self,_,line):
        try:
            fields = line.split(",")
            if len(fields) == 9:
                miner_address = fields[2]
                size = int(fields[4])
                yield (miner_address, size)
        except:
            pass

    def reducer(self, word, values):
        total = sum(values)
        yield(word,total)

if __name__ == '__main__':
    partb.run()
#python parta.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions
#hadoop fs -copyToLocal out
#hadoop fs -rm -r out
#hadoop fs -ls out
#python Partc_job_1.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/blocks
```

The Mapper:

- I check if the file is blocks from the previous job by checking if the length of fields is 9.
- Then I extract fields[2] to get miner address and fields[4] to get the size of blocks mined. Then I yield both of these values.

The reducer:

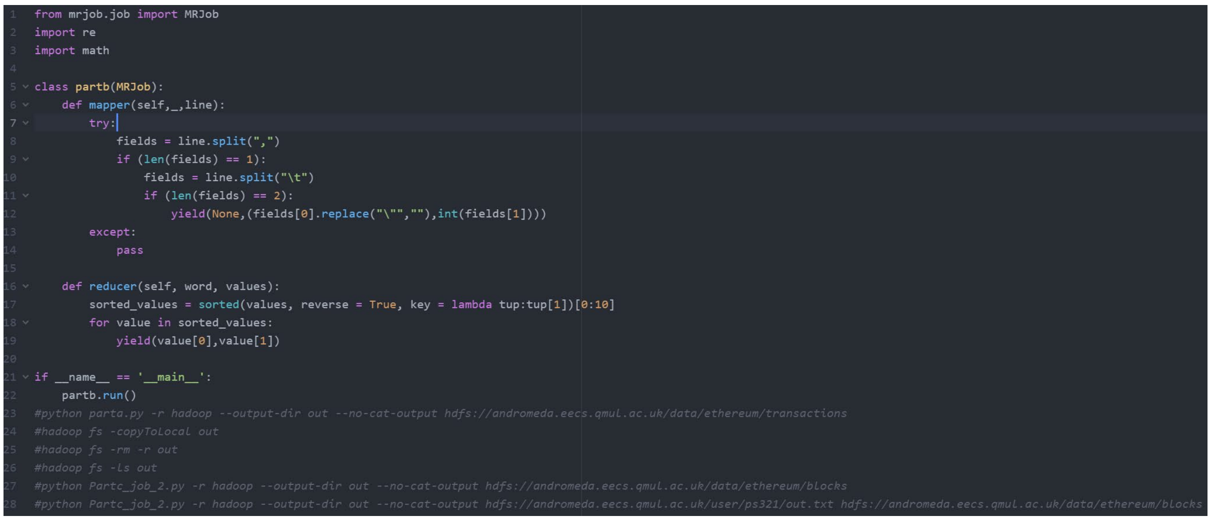
- I use the built in python sum() function to sum all the miners blocks and put it in the total variable.
- Then I yield the word parameter which is the miner address and the total which is the total number of blocks mined by that miner address.

Command to run after environment is set up:python Partc_job_1.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/blocks

File name: PartC/Partc_Job1.py

Part C(Job2): In this job I find the top 10 miners using the data from the previous job.

Before starting this job, I used the Hadoop get merge command to get a merged output file from job1 then I used copy to Hadoop filesystem command to copy the merged file back into Hadoop file system ready to perform a top 10 job on it.



```python
from mrjob.job import MRJob
import re
import math

class partb(MRJob):
    def mapper(self,_,line):
        try:
            fields = line.split(",")
            if (len(fields) == 1):
                fields = line.split("\t")
                if (len(fields) == 2):
                    yield(None,(fields[0].replace("\"",""),int(fields[1])))
        except:
            pass

    def reducer(self, word, values):
        sorted_values = sorted(values, reverse = True, key = lambda tup:tup[1])[0:10]
        for value in sorted_values:
            yield(value[0],value[1])

if __name__ == '__main__':
    partb.run()
#python parta.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions
#hadoop fs -copyToLocal out
#hadoop fs -rm -r out
#hadoop fs -ls out
#python Partc_job_2.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/blocks
#python Partc_job_2.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/user/ps321/out.txt hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/blocks
```

The Mapper:

- I check if the file is blocks from the previous job by checking if the length of fields is 1. If it's the case I split the record of the dataset by "/t" as the out.txt file has whitespaces between the values.
- As I have now split the dataset, I check if the dataset is out.txt by checking if the length is 2 and if it is I yield the miner address with removed backslashes with replaced commas and the total blocks mined and a None type, this none type is because of the sorting taking place in the reducer.

The reducer:

- I use the built in python sorted() function, where it takes the values parameter as the data to sort. I have reverse = True to ensure the data is sorted in reverse. I have a key equal to the lambda function which would return the position[1] of the values as we want to sort by transaction value. I also include a [0:10] so the command is only run 10 times.
- Finally, I iterate over sorted values yielding values[0] and values[1] which is the miners address and the blocks value of the top 10 miners, which gives us the top 10 most active miners.

Command to run after environment is set up: python Partc_job_2.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/user/ps321/out.txt hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/blocks

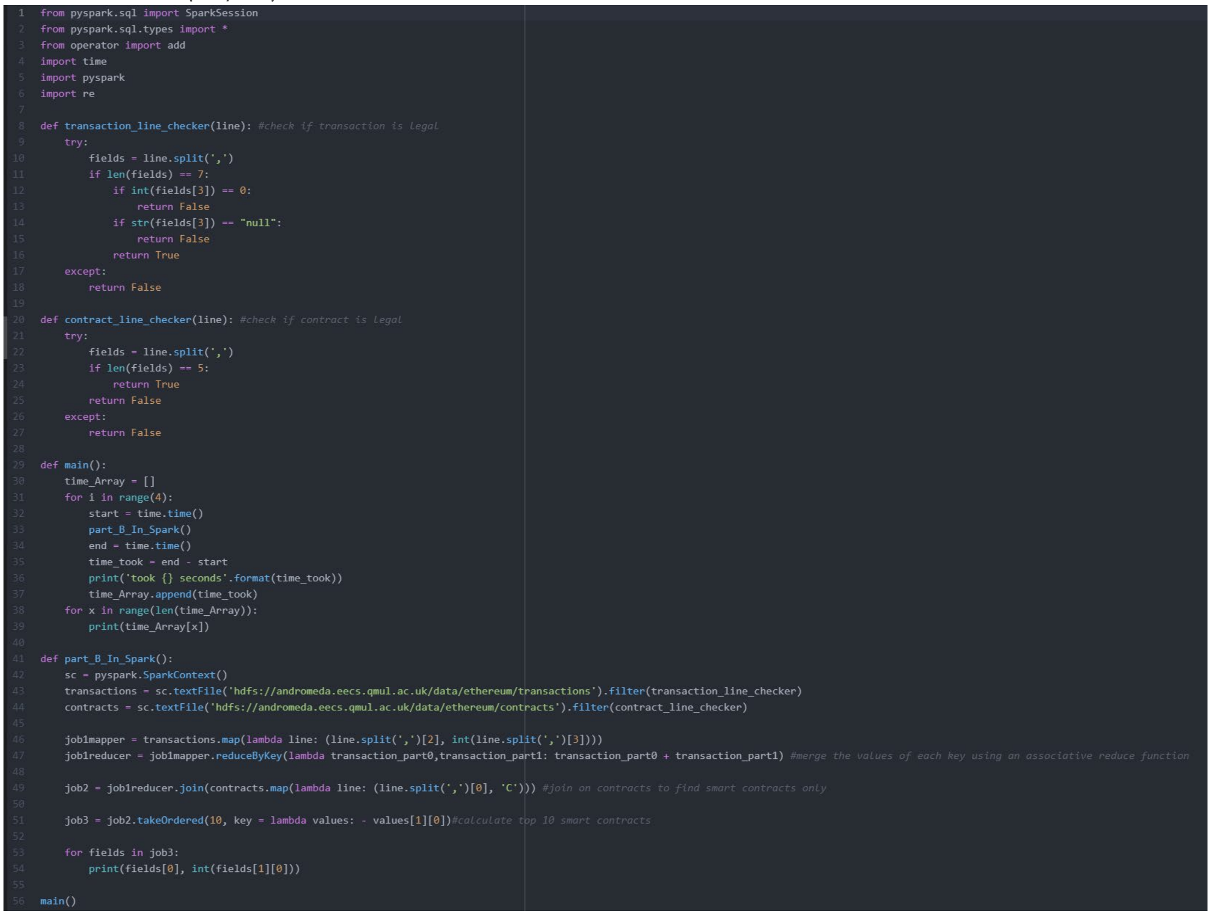
File name: PartC/Partc_Job2.py

Output file: PartC/partc_out.txt

Part D:

Comparative Evaluation Reimplement Part B in Spark (if your original was MRJob, or vice versa). How does it run in comparison? Keep in mind that to get representative results you will have to run the job multiple times, and report median/average results.

Can you explain the reason for these results? What framework seems more appropriate for this task? (10/50)



```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from operator import add
import time
import pyspark
import re

def transaction_line_checker(line): #check if transaction is legal
    try:
        fields = line.split(',')
        if len(fields) == 7:
            if int(fields[3]) == 0:
                return False
            if str(fields[3]) == "null":
                return False
            return True
    except:
        return False

def contract_line_checker(line): #check if contract is legal
    try:
        fields = line.split(',')
        if len(fields) == 5:
            return True
        return False
    except:
        return False

def main():
    time_Array = []
    for i in range(4):
        start = time.time()
        part_B_In_Spark()
        end = time.time()
        time_took = end - start
        print('took {} seconds'.format(time_took))
        time_Array.append(time_took)
    for x in range(len(time_Array)):
        print(time_Array[x])

def part_B_In_Spark():
    sc = pyspark.SparkContext()
    transactions = sc.textFile('hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions').filter(transaction_line_checker)
    contracts = sc.textFile('hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/contracts').filter(contract_line_checker)

    job1mapper = transactions.map(lambda line: (line.split(',')[2], int(line.split(',')[3])))
    job1reducer = job1mapper.reduceByKey(lambda transaction_part0,transaction_part1: transaction_part0 + transaction_part1) #merge the values of each key using an associative reduce function

    job2 = job1reducer.join(contracts.map(lambda line: (line.split(',')[0], 'C'))) #join on contracts to find smart contracts only

    job3 = job2.takeOrdered(10, key = lambda values: - values[1][0])#calculate top 10 smart contracts

    for fields in job3:
        print(fields[0], int(fields[1][0]))

main()
```

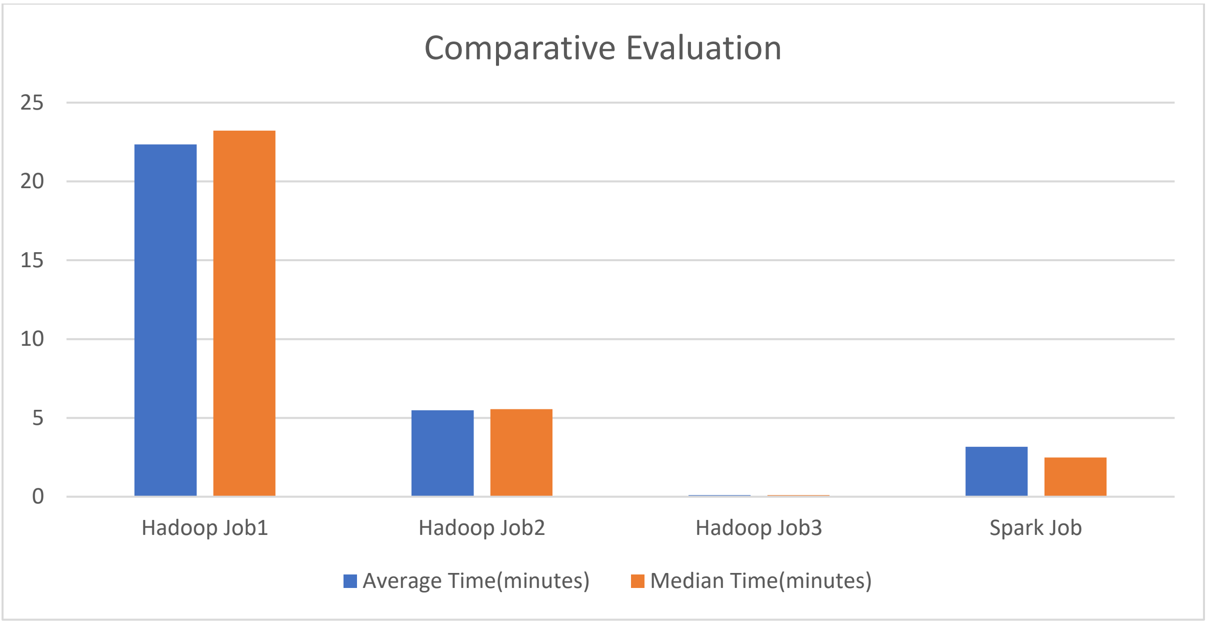
I make 3 helper functions:

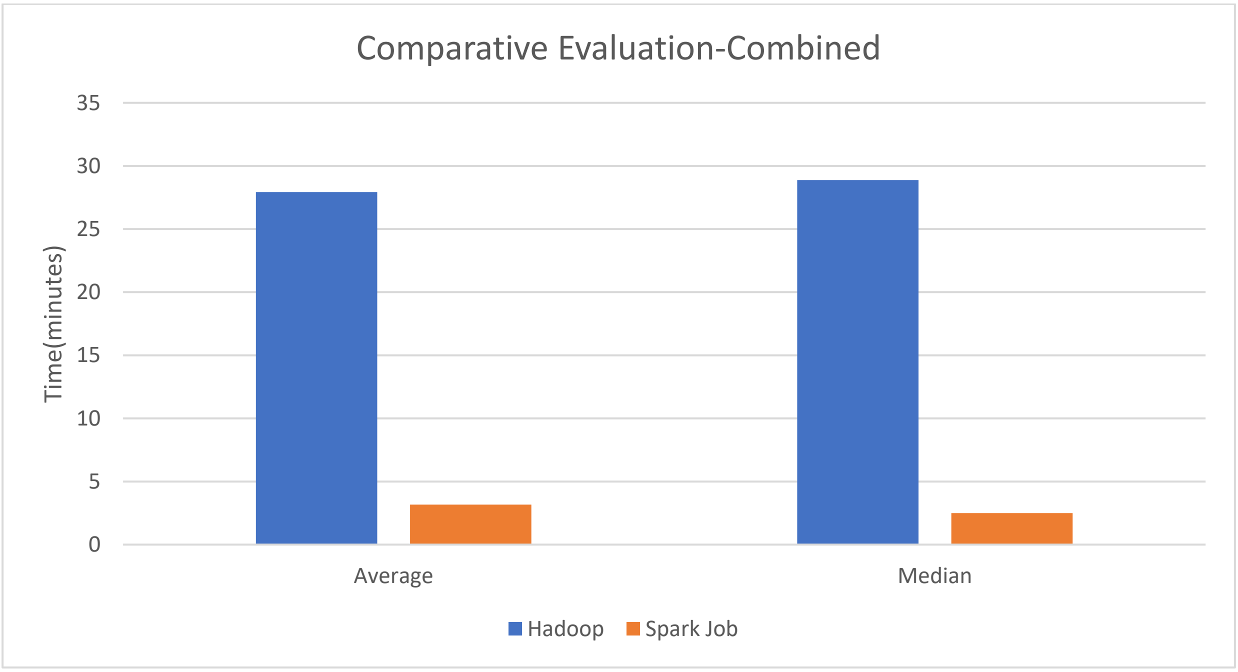
- transaction_line_checker() and contract_line_checker() functions take fields in the two dataset and check if the lines are not malformed. They do this by returning True of False depending on the line and if it returns false that specific line is then taken out of the copy of the dataset.
- I then use the .map() function to split the records data by commas and get aggregates of address and transaction vales by extracting them first from the fields[2] and fields[3] from transactions dataset using a lambda function.
- Then I get the actual aggregates using a reduceByKey() function which uses lambda to sum all transaction values for that specific address by using values of each key as an associative reduce function.
- Then I perform a join on the previous aggregate by contracts to user contracts and find smart contracts only. I do this by using a lambda function.
- Then I find the top 10 smart contracts by using the takeOrdered() function which takes the number of records to output, which I wrote as 10 then I use a lambda function again to get the values in reverse order.
- Finally, I loop through the fields in the top 10 output to print the address and the transaction value.
- This entire function is then called between two time.time() statements in the main method to gain the time taken to compute the function.
- This process is repeated 5 times for this function to calculate an average.

The job can be run using the following command : python partB(spark).py

File name: PartD/Comparative Analysis/partB(spark).py

From part B Job1,job2,job3 is run 5 times each to get an average time and median time of map reduce:



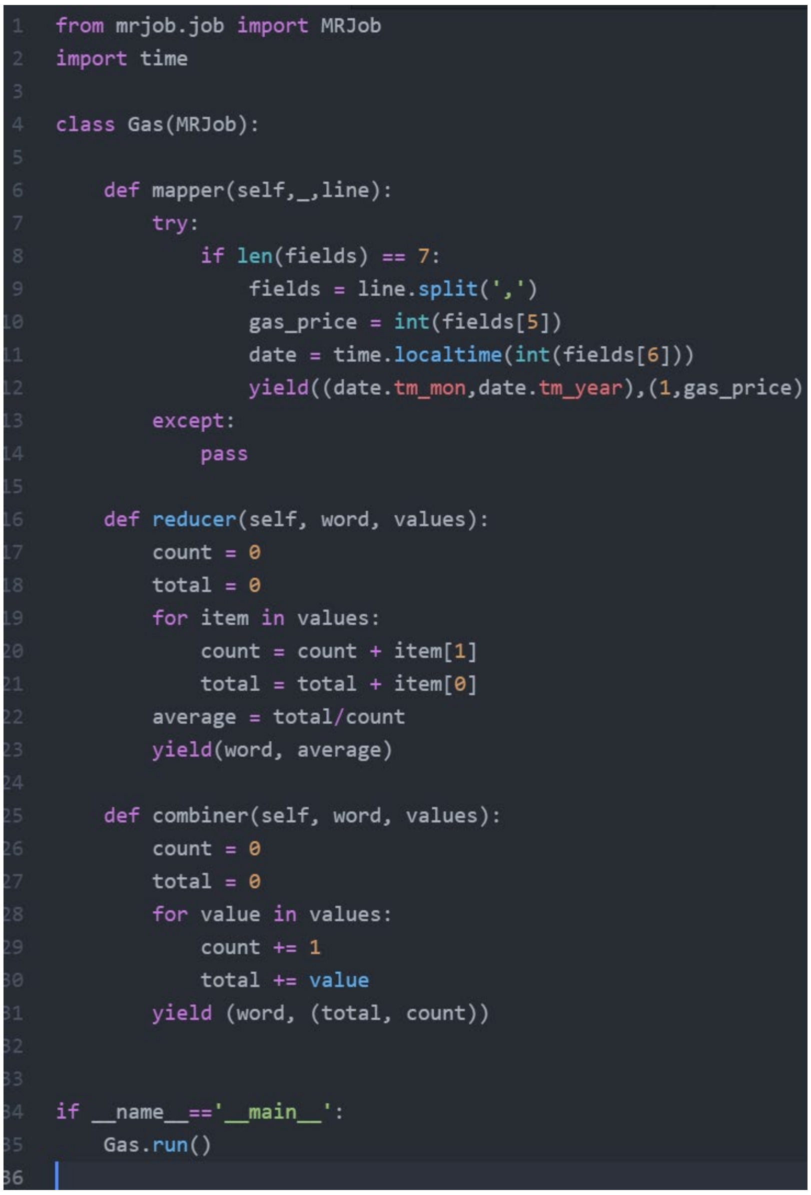


- Hadoop jobs take significant more time to run then spark jobs.
- Job1 using Hadoop takes the longest at an average of 22 minutes.
- The only Hadoop job that runs faster than spark is Hadoop job3, but even that data is not significant as the spark job does all 3 Hadoop jobs in one go.
- So, the total average time for the Hadoop job is 28 minutes and 23 seconds. This was during the time the Hadoop file system was not under too much load.

- Spark is faster because it uses random access memory (RAM) instead of reading and writing data to disks. Hadoop stores data on multiple sources and processes it in batches by MapReduce.
- Hadoop is a lower cost solution as it uses disk storage and not RAM, it currently costs 3.71 GBP per GB of RAM and 0.023 GBP per GB of Hard Disk Storage. (HumanProgress.org, 2015)
- This Job is better suited for the Spark framework, as although the cost will be higher it would be better in a workplace environment where the data scientist needs to get the results of the job to run another job on the dataset.

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. (10/50)

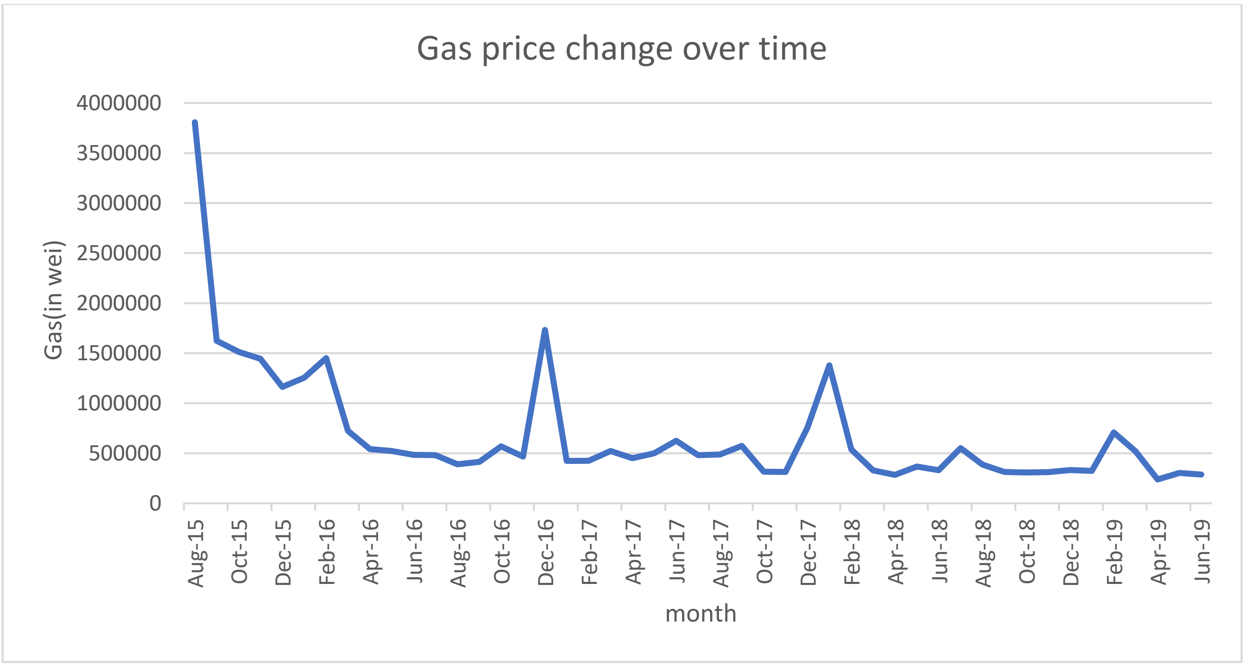
“Gas fees are payments made by users to compensate for the computing energy required to process and validate transactions on the Ethereum blockchain. "Gas limit" refers to the maximum amount of gas (or energy) that you're willing to spend on a particular transaction. A higher gas limit means that you must do more work to execute a transaction using ETH or a smart contract” (FRANKENFIELD, 2021).



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

class Gas(MRJob):

    def mapper(self,_,line):
        try:
            if len(fields) == 7:
                fields = line.split(',')
                gas_price = int(fields[5])
                date = time.localtime(int(fields[6]))
                yield((date.tm_mon,date.tm_year),(1,gas_price)
        except:
            pass

    def reducer(self, word, values):
        count = 0
        total = 0
        for item in values:
            count = count + item[1]
            total = total + item[0]
        average = total/count
        yield(word, average)

    def combiner(self, word, values):
        count = 0
        total = 0
        for value in values:
            count += 1
            total += value
        yield (word, (total, count))


if __name__=='__main__':
    Gas.run()

```

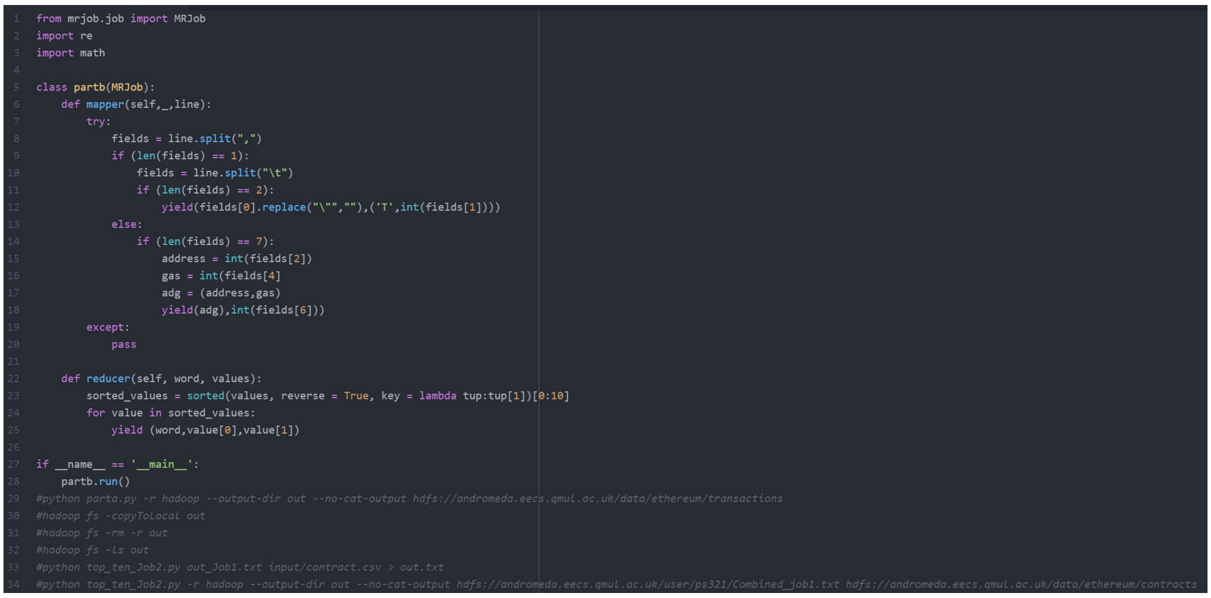
File Name: PartD/Gas Guzzlers/Gas_Job1.py

Firstly, I ran job1 to get the average price per gas for every month in the dataset. I extracted the date and gas price and worked out the average gas price per month. This was the graph that was produced from the code:



- The plot shows at the start of the Dataset the gas prices were really high and then from August 2015 to April 2016 the price decreased and remained stagnant apart from two peaks in December 2016 and January 2018.
- After the high gas prices at the start of the dataset the gas price has never peaked over that price.

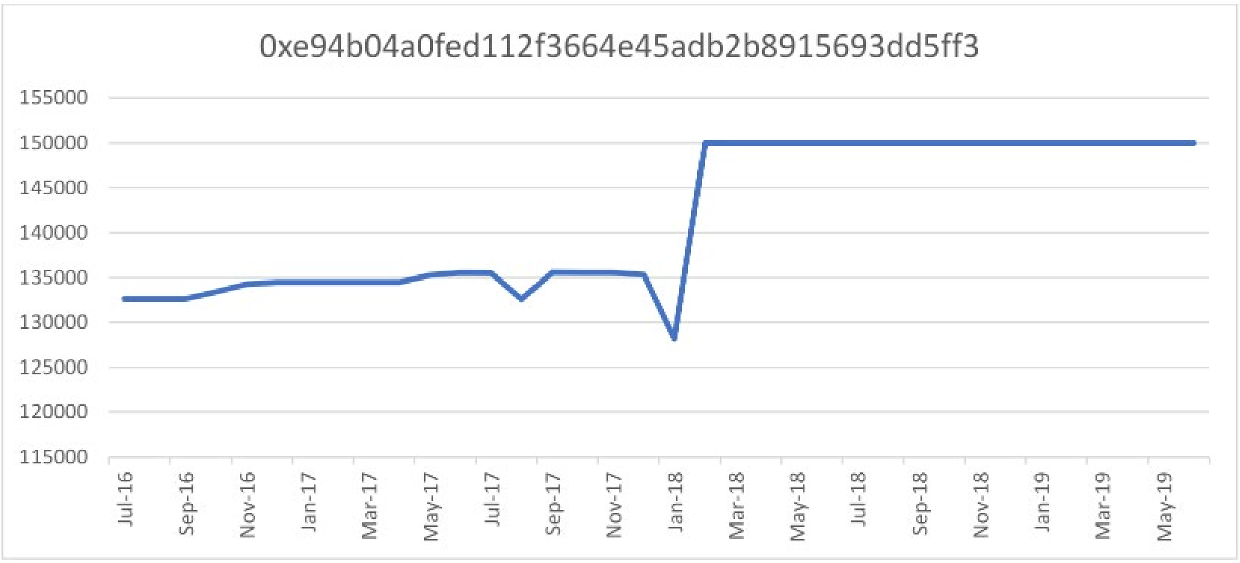
Then, I ran job2 to get a replication join between the top 10 smart contract addresses to the transaction's dataset, to see if contracts became complicated requiring more gas over time. I got the addresses of the top 10 smart contracts from part B.



```python
from mrjob.job import MRJob
import re
import math

class partb(MRJob):
    def mapper(self,_,line):
        try:
            fields = line.split(",")
            if (len(fields) == 1):
                fields = line.split("\t")
                if (len(fields) == 2):
                    yield(fields[0].replace("\"",""),('T',int(fields[1])))
            else:
                if (len(fields) == 7):
                    address = int(fields[2])
                    gas = int(fields[4]
                    adg = (address,gas)
                    yield(adg),int(fields[6]))
        except:
            pass

    def reducer(self, word, values):
        sorted_values = sorted(values, reverse = True, key = lambda tup:tup[1])[0:10]
        for value in sorted_values:
            yield (word,value[0],value[1])

if __name__ == '__main__':
    partb.run()
#python parta.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions
#hadoop fs -copyToLocal out
#hadoop fs -rm -r out
#hadoop fs -ls out
#python top_ten_Job2.py out_Job1.txt input/contract.csv > out.txt
#python top_ten_Job2.py -r hadoop --output-dir out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/user/ps321/Combined_job1.txt hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/contracts
```

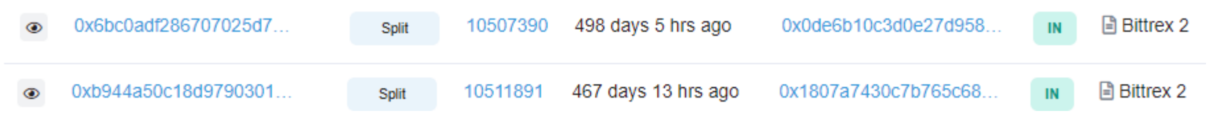
File Name: PartD/Gas Guzzlers/Gas_Job2.py

The graph that stood out the most from the ten others was:

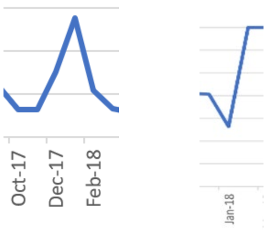


Average gas consumed by "Bittrex_2" in months

- After some research online the address: 0xe94b04a0fed112f3664e45adb2b8915693dd5ff3 refers to the smart contract "Bittrex_2" which is a layer provided between users and the Ethereum network to make fast transactions with low transaction fees (Bittrex Team, 2021).
- It is also seen that from the first transaction to the last transaction made to the Bittrex_2 smart contract there was a change in Ethereum gas prices (Anon., 2021).



👁	0x6bc0adf286707025d7…	Split	10507390	498 days 5 hrs ago	0x0de6b10c3d0e27d958…	IN	Bittrex 2
👁	0xb944a50c18d9790301…	Split	10511891	467 days 13 hrs ago	0x1807a7430c7b765c68…	IN	Bittrex 2

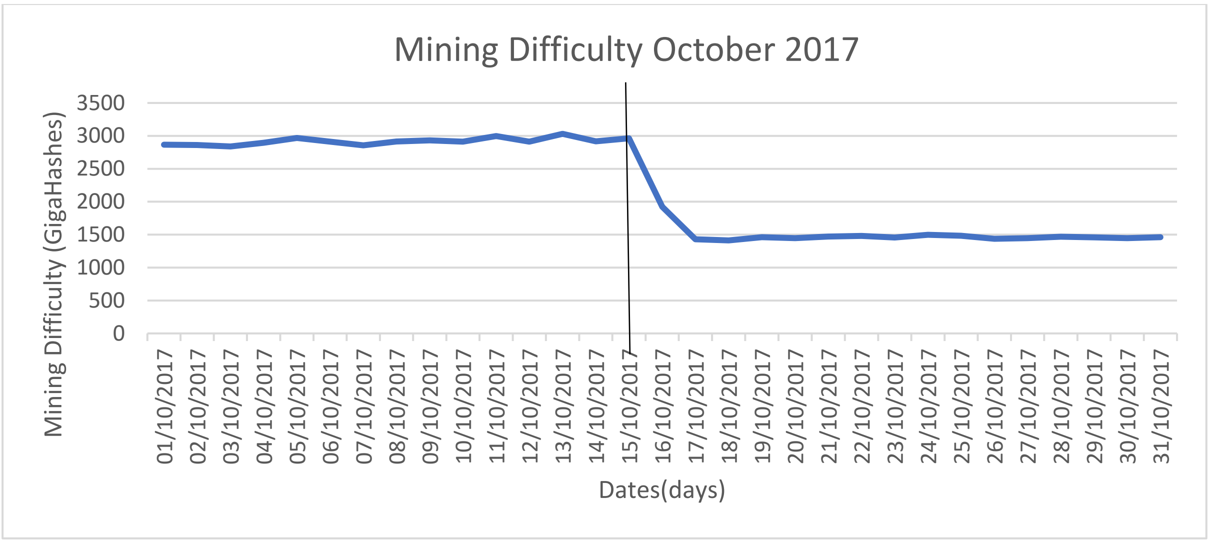


- This implies that "Bittrex_2" caused gas prices to rise up to three times its original value in January 2018.

Fork the Chain: There have been several forks of Ethereum in the past. Identify one or more of these and see what effect it had on price and general usage. For example, did a price surge/plummet occur and who profited most from this? (10/50)

I Decided to research on the Byzantium fork that happened on October 16th, 2017:

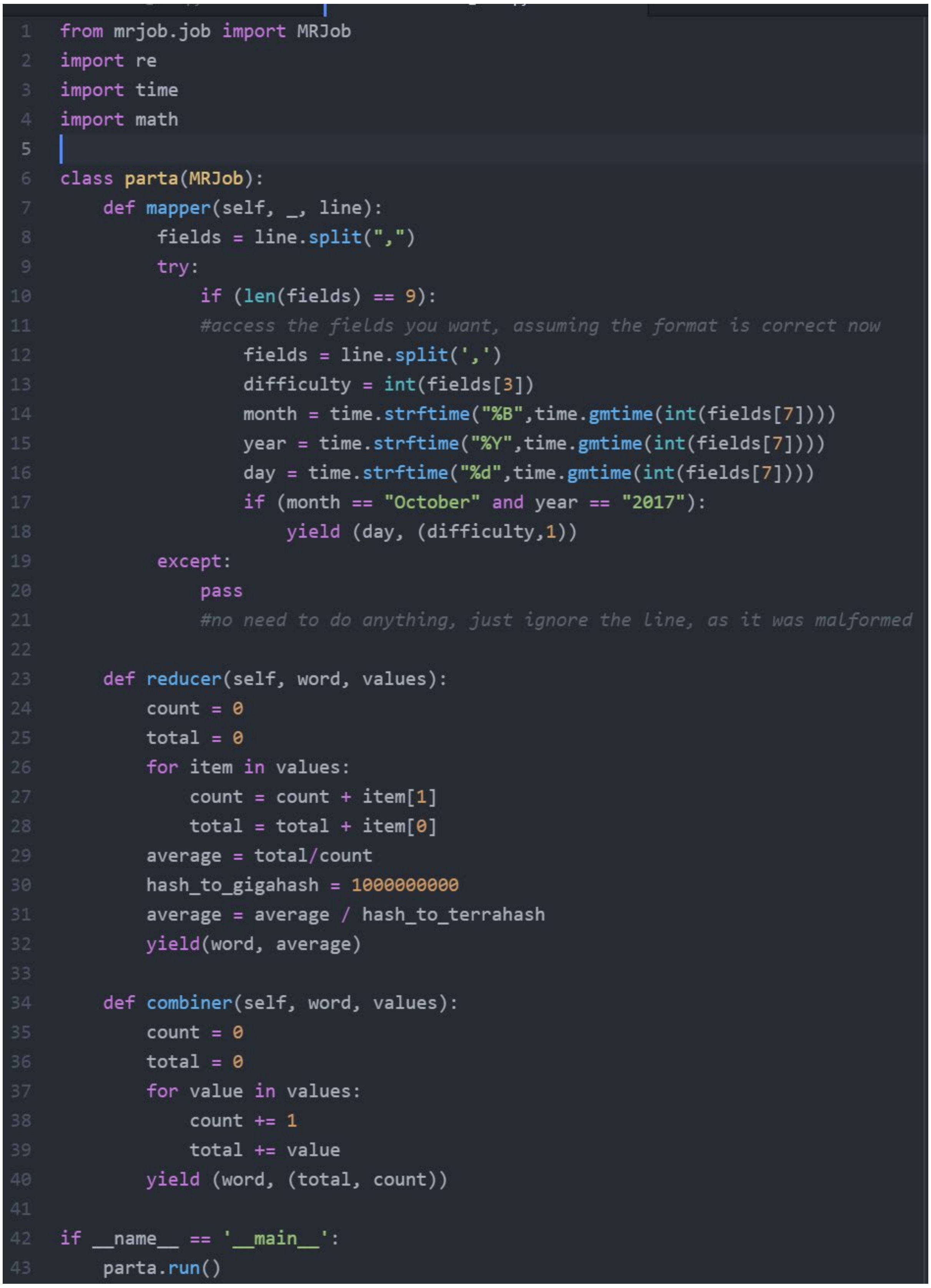
- Firstly, I looked at the whole month of October 2017 on gas prices, and the activity of the top 10 contracts and mining difficulty.
- I found a trend in mining difficulty after the fork.



- The black line denotes 16th October the day the fork happened. The mining difficulty per block was greatly decreased on the day it happened.
- When looking at the Ethereum price a month after the fork happened, it was increasing at a rapid pace. This was because this fork added certain cryptographic methods to allow better scaling of the currency as well as further securing transactions.
- After I found out about the mining difficulty going down in the graph, I did some research behind the Byzantium fork and found out that this fork has essentially delayed a difficulty bomb by a year, where difficulty bomb refers to “refers to the increasing difficulty level of puzzles in the mining algorithm used to reward miners with ether on its blockchain” (Sharma, 2019). This means that Ethereum miners profited the most from this as they were able to mine more Ethereum.

The following code shows how I got the results for mining difficulty in October 2017:

- I Extract month, year, and day from the dataset.
- I yielded only if the days falls in October and 2017 as this was the month where the fork happened.
- I found the average mining difficulty in hashes and converted it into Giga hashes by dividing the hashes by 1000000000.
- I then yield the (day, month, year) and average Giga hash.



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

class parta(MRJob):
    def mapper(self, _, line):
         fields = line.split(",")
         try:
             if (len(fields) == 9):
             #access the fields you want, assuming the format is correct now
                 fields = line.split(',')
                 difficulty = int(fields[3])
                 month = time.strftime("%B",time.gmtime(int(fields[7])))
                 year = time.strftime("%Y",time.gmtime(int(fields[7])))
                 day = time.strftime("%d",time.gmtime(int(fields[7])))
                 if (month == "October" and year == "2017"):
                     yield (day, (difficulty,1))
         except:
             pass
             #no need to do anything, just ignore the line, as it was malformed

    def reducer(self, word, values):
        count = 0
        total = 0
        for item in values:
            count = count + item[1]
            total = total + item[0]
        average = total/count
        hash_to_gigahash = 1000000000
        average = average / hash_to_terrahash
        yield(word, average)

    def combiner(self, word, values):
        count = 0
        total = 0
        for value in values:
            count += 1
            total += value
        yield (word, (total, count))

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

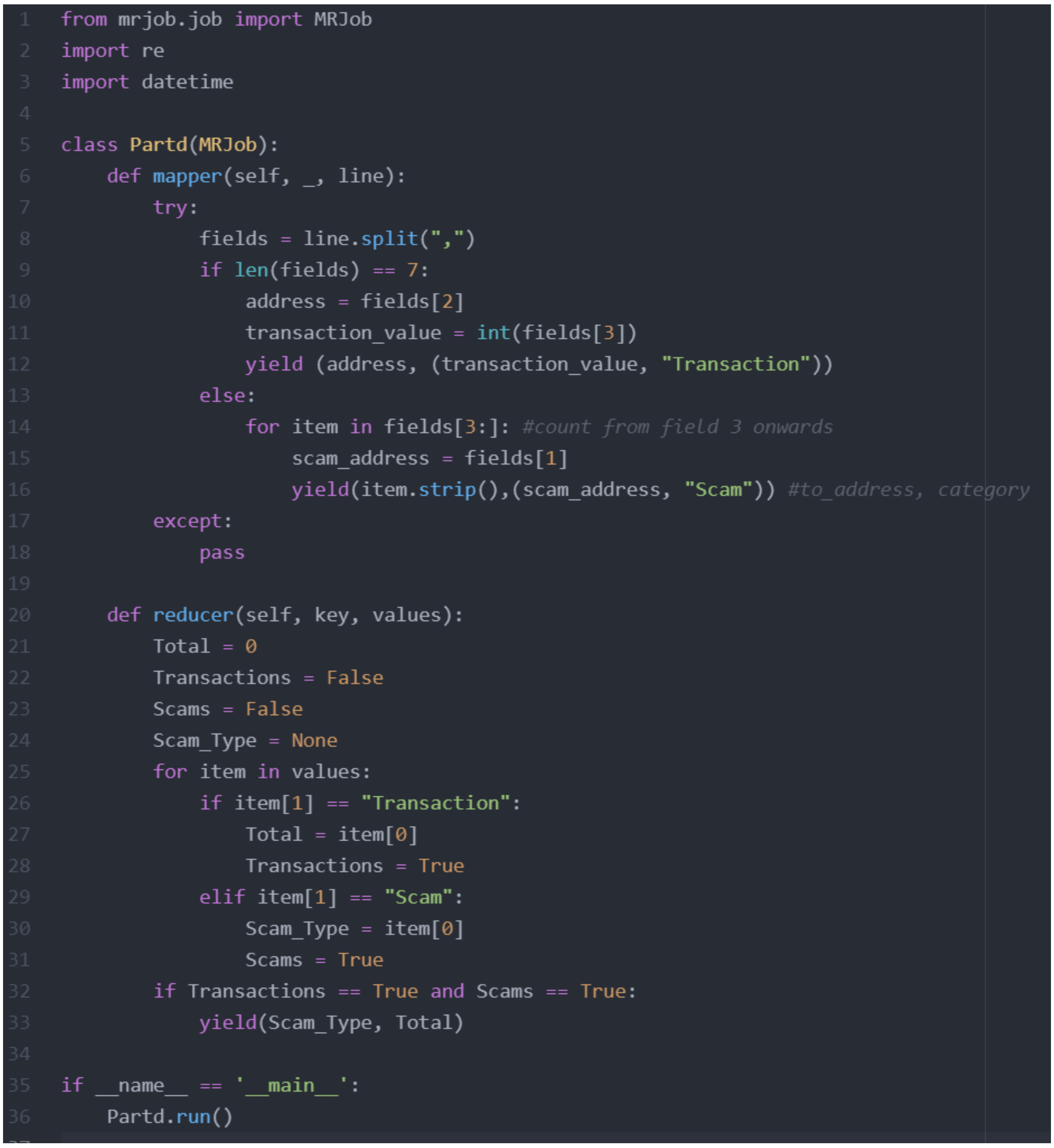
File Name: PartD/Fork/ fork_chain.py

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? (15/50)

What is the most lucrative form of scam?

I am using the json parser received from the module announcemnets to extract the fields I need from the scams.json.

Job1: In this job I find out how much wei was stolen in each scam.



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

class Partd(MRJob):
    def mapper(self, _, line):
        try:
            fields = line.split(",")
            if len(fields) == 7:
                address = fields[2]
                transaction_value = int(fields[3])
                yield (address, (transaction_value, "Transaction"))
            else:
                for item in fields[3:]: #count from field 3 onwards
                    scam_address = fields[1]
                    yield(item.strip(),(scam_address, "Scam")) #to_address, category
        except:
            pass

    def reducer(self, key, values):
        Total = 0
        Transactions = False
        Scams = False
        Scam_Type = None
        for item in values:
            if item[1] == "Transaction":
                Total = item[0]
                Transactions = True
            elif item[1] == "Scam":
                Scam_Type = item[0]
                Scams = True
        if Transactions == True and Scams == True:
            yield(Scam_Type, Total)

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

File Name: PartD/Scam Analysis/Lucrative_Job1.py

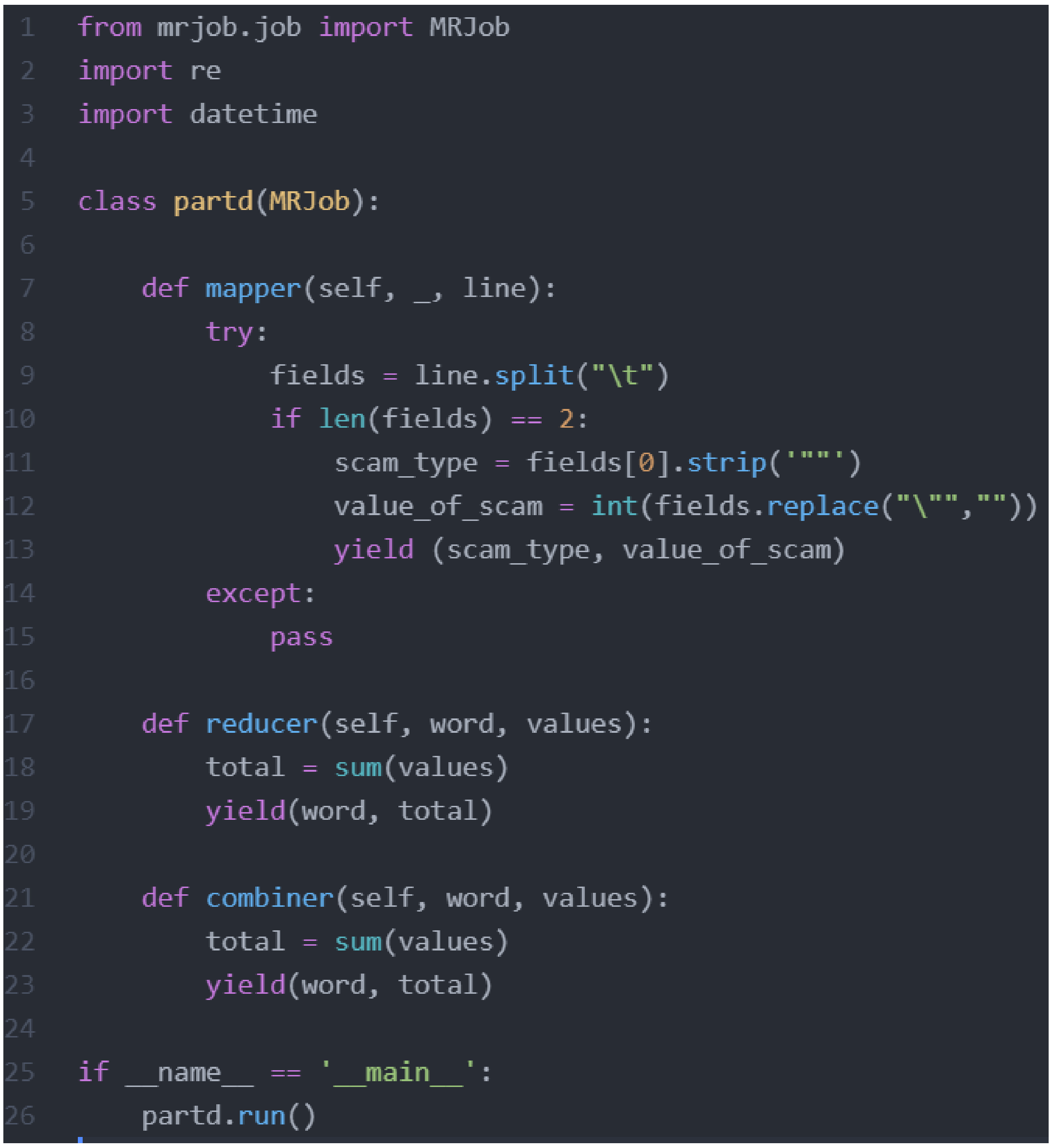
The Mapper:

- Firstly, I split the data and check if the fields are legal, yielding transaction value and “Transaction” if they are legal.
- If they are not legal, its most likely a scam. So, I loop through the fields starting from fields 3 onwards as that is where the scam category is.
- When yielding I strip the data to ensure there are no spaces left from the json file, I also yield the scam addresses and “Scam”.

The Reducer:

- I then loop through all the yielded data checking if there are any Scams and Transactions to any said Scams.
- I then yield the scam type and the total amount of Wei that was scammed.

Job2: I aggregate the amount of wei stolen for each type of scam, telling me which scam was the most lucrative.



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

class partd(MRJob):

    def mapper(self, _, line):
        try:
            fields = line.split("\t")
            if len(fields) == 2:
                scam_type = fields[0].strip('""')
                value_of_scam = int(fields.replace("\"",""))
                yield (scam_type, value_of_scam)
        except:
            pass

    def reducer(self, word, values):
        total = sum(values)
        yield(word, total)

    def combiner(self, word, values):
        total = sum(values)
        yield(word, total)

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

File Name: PartD/Scam Analysis/Lucrative_Job2.py

The Mapper:

- I split the record of the dataset by "/t" as the out.txt file has whitespaces between the values.
- I then check if the record is of size 2 to check for any malformed lines.
- I then format the data by stripping any quotation marks and removing backslashes.
- I yield the scam type and value of scam.

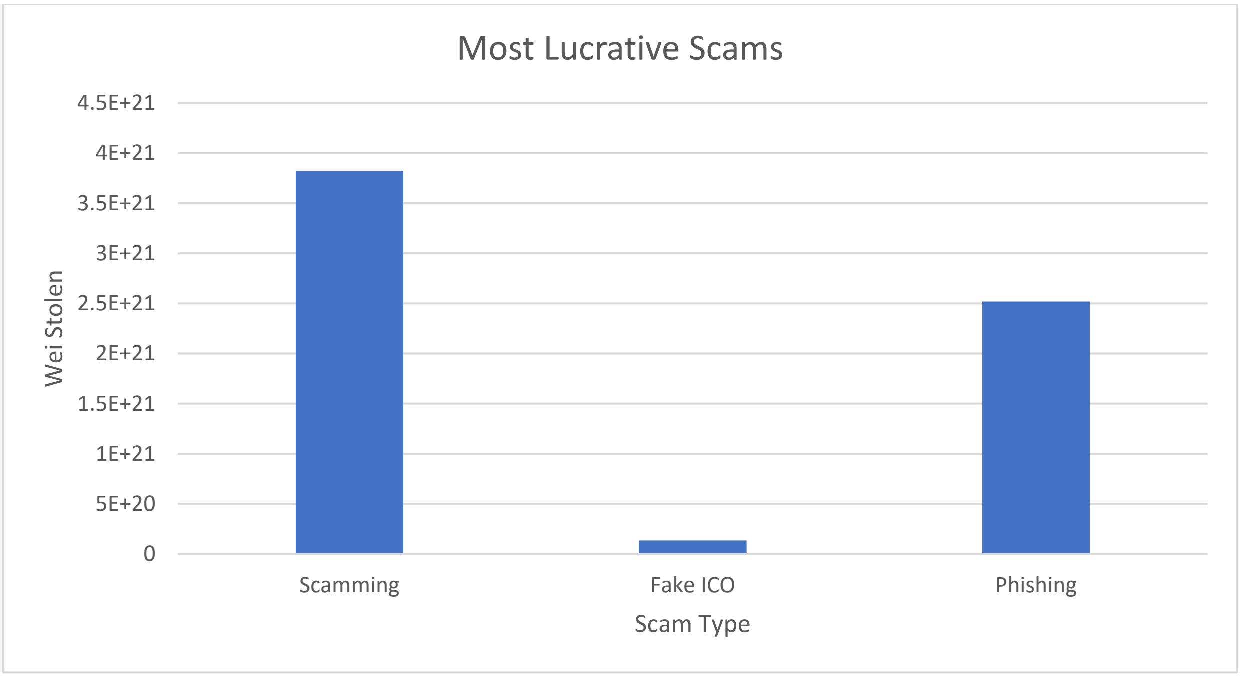
The Combiner:

- I combine all key value pairs here for the reducer to aggregate.

The Reducer:

- I aggregate all the value_of_scam for each scam type.
- Finally yielding the scam type and the total number of money stolen by that specific scam type.

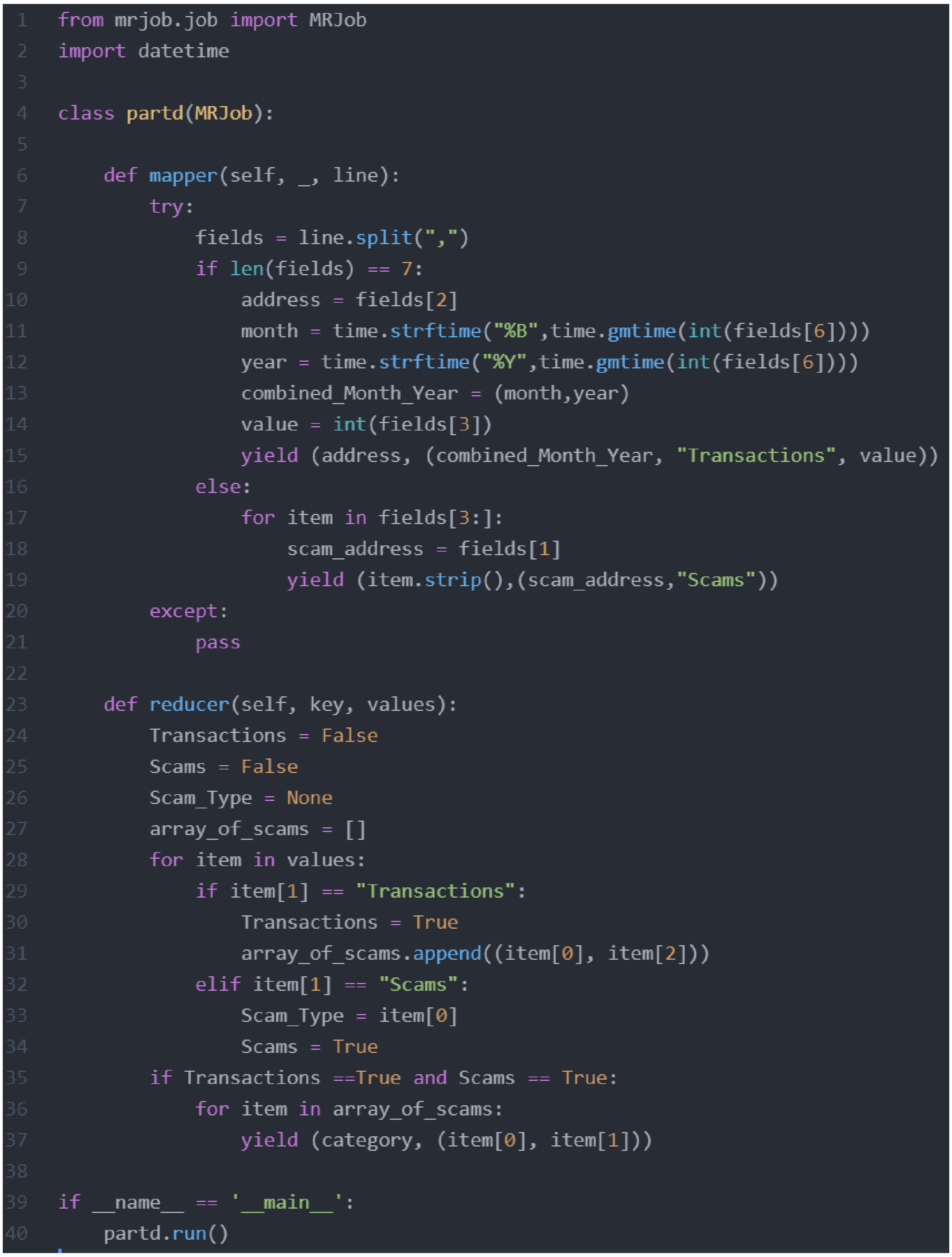
The following graph was made from the resultant data:



- This data tells us that the most lucrative scams are "Scamming", this is a umbrella term used for the common types of scams that occur such as posing as imposters.
- This data also tells us that Fake ICO scams were the least lucrative as the other scams stole a lot more wei.
- Phishing scams were not as lucrative as "Scamming" but were a lot more lucrative then Fake ICO.
- The reason why Fake ICO scams are not as lucrative is that you generally have to have a lot of capital to fall into a Fake ICO scam.

How does this change throughout time?

Job3: In this job I find out how much wei was stolen in each scam, similar to job1. However, in Job3 I also find out the date of the scam too.



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

class partd(MRJob):

    def mapper(self, _, line):
        try:
            fields = line.split(",")
            if len(fields) == 7:
                address = fields[2]
                month = time.strftime("%B",time.gmtime(int(fields[6])))
                year = time.strftime("%Y",time.gmtime(int(fields[6])))
                combined_Month_Year = (month,year)
                value = int(fields[3])
                yield (address, (combined_Month_Year, "Transactions", value))
            else:
                for item in fields[3:]:
                    scam_address = fields[1]
                    yield (item.strip(),(scam_address,"Scams"))
        except:
            pass

    def reducer(self, key, values):
        Transactions = False
        Scams = False
        Scam_Type = None
        array_of_scams = []
        for item in values:
            if item[1] == "Transactions":
                Transactions = True
                array_of_scams.append((item[0], item[2]))
            elif item[1] == "Scams":
                Scam_Type = item[0]
                Scams = True
        if Transactions ==True and Scams == True:
            for item in array_of_scams:
                yield (category, (item[0], item[1]))

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

File Name: Lucrative_Job3.py

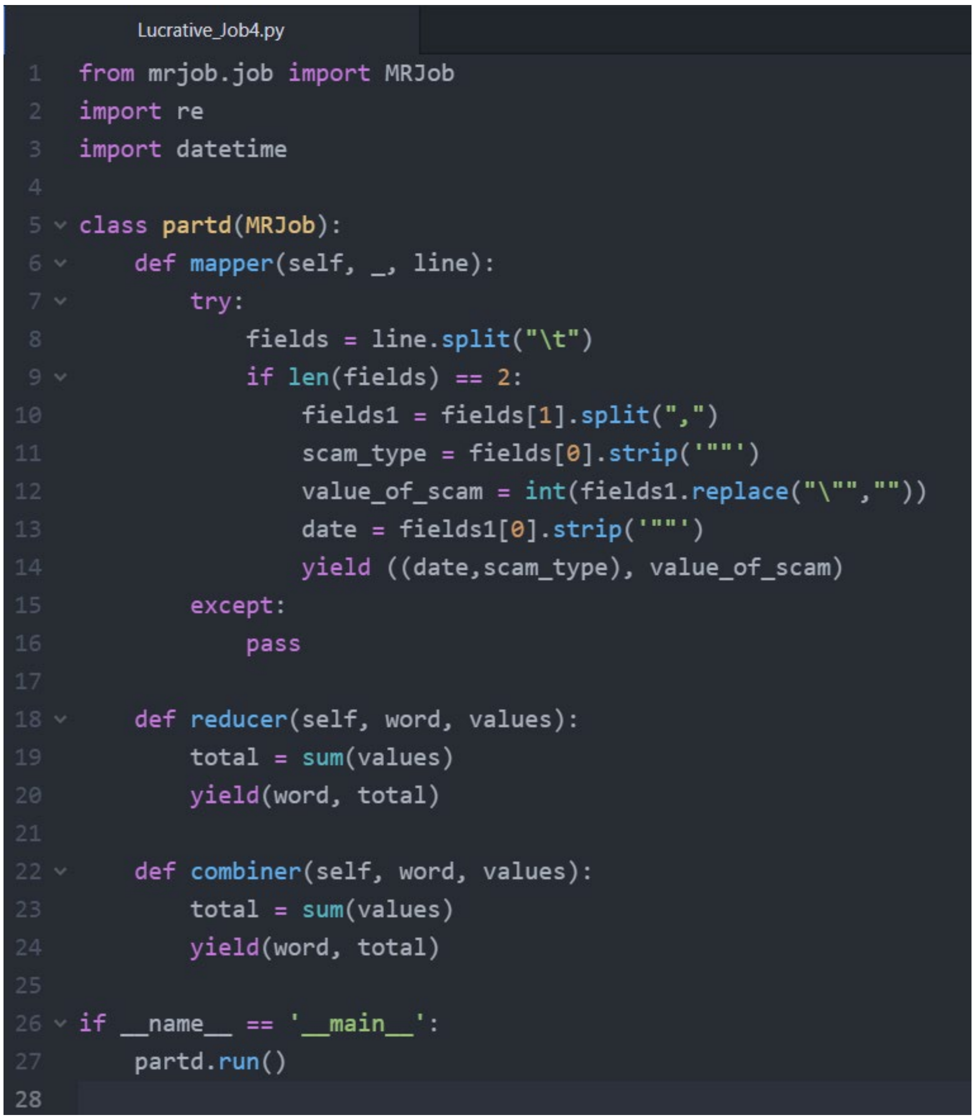
The Mapper:

- Firstly, I split the data and check if the fields are legal, yielding transaction value, date and “Transaction” if they are legal.
- If they are not legal, its most likely a scam. So, I loop through the fields starting from fields 3 onwards as that is where the scam category is.
- When yielding I strip the data to ensure there are no spaces left from the json file, I also yield the scam addresses and “Scam”.

The Reducer:

- I then loop through all the yielded data checking if there are any Scams and Transactions to any said Scams.
- While looping through the data I add the date and the value into an array for storage.
- Then I loop through the array yielding the scam type and the total amount of Wei that was scammed and the date it happened.

Job4: I aggregate the amount of wei stolen for each type of scam over months and years, telling me which scam was the most lucrative over time.



Lucrative_Job4.py

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

class partd(MRJob):
    def mapper(self, _, line):
        try:
            fields = line.split("\t")
            if len(fields) == 2:
                fields1 = fields[1].split(",")
                scam_type = fields[0].strip('"""')
                value_of_scam = int(fields1.replace("\"",""))
                date = fields1[0].strip('"""')
                yield ((date,scam_type), value_of_scam)
        except:
            pass

    def reducer(self, word, values):
        total = sum(values)
        yield(word, total)

    def combiner(self, word, values):
        total = sum(values)
        yield(word, total)

if __name__ == '__main__':
    partd.run()

```

File Name: PartD/Scam Analysis/Lucrative_Job4.py

The Mapper:

- I split the record of the dataset by "/t" as the out.txt file has whitespaces between the values.
- I then check if the record is of size 2 to check for any malformed lines.
- Now I split the fields between the values dates and total amount of wei stolen. This is because the last Job has two bits of data yield, however 1 bit of data was in this format: (data, data).
- I then format the data by stripping any quotation marks and removing backslashes.
- I yield the scam type and value of scam and date of scam.

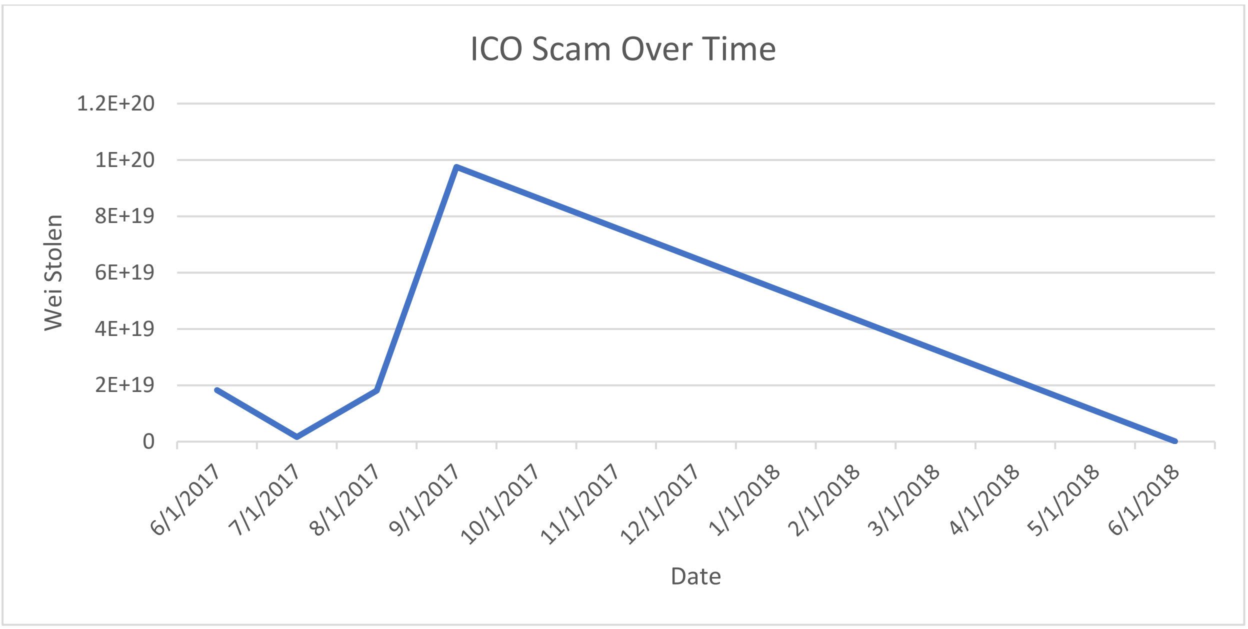
The Combiner:

- I combine all key value pairs here for the reducer to aggregate.

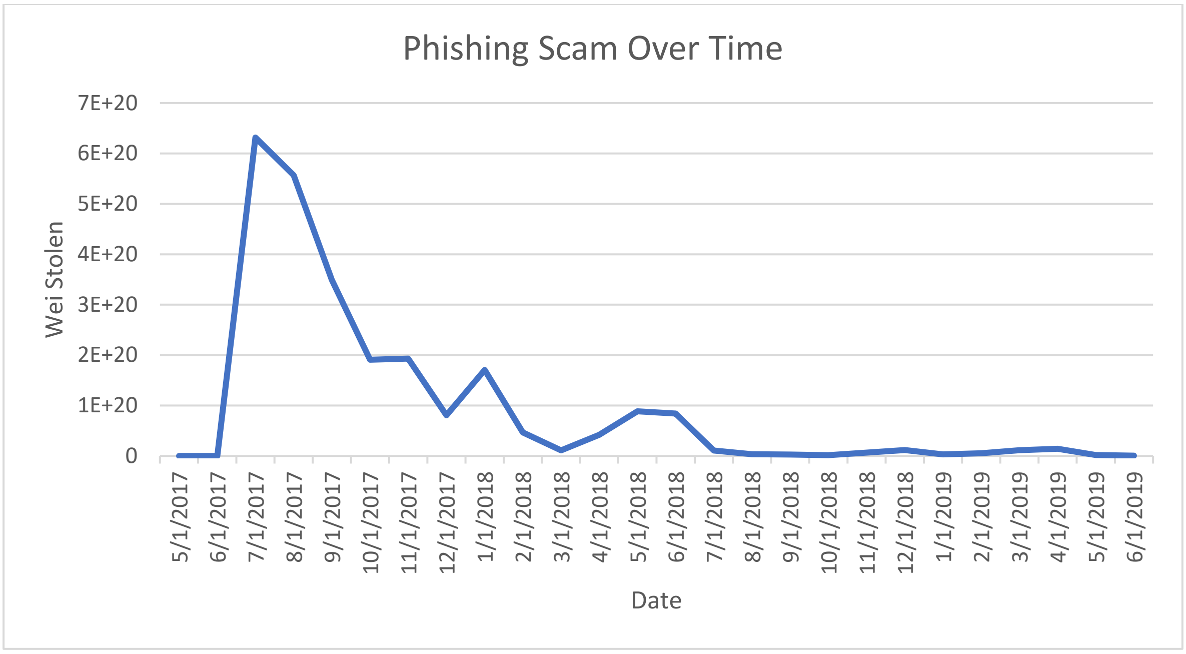
The Reducer:

- I aggregate all the value_of_scam for each scam type and date.
- Finally yielding the scam type and the total number of money stolen by that specific scam type for months where that scam type took place.

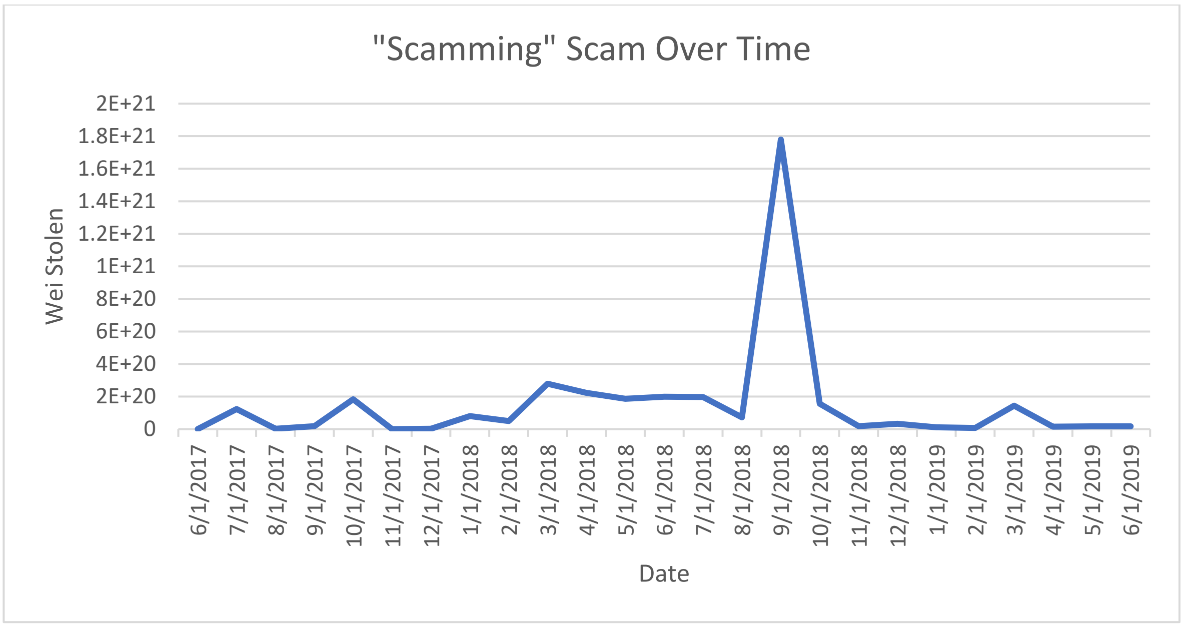
The Graphs Produced:



- This graph shows how Fake ICO scams evolved over time.
- The Scam gained traction between August 2017 and September 2017, thereafter the scam plummeted.
- There is very little empirical data on this type of scam.



- This graph shows the phishing scam over time, The phishing scam increased exponentially between June 2017 and July 2017. It also peaked in July 2017.
- The scam fell off after July 2017 rising slightly in January 2018 and May 2018 before falling off(going inactive).
- This scam was most popular during 2017.



- The Scamming Scam increased exponentially between August 2018 and September 2018, peaking in September 2018.
- The scam fell off after this and didn’t recover.
- This scam peaked higher than any other scam.

Bringing it all together:

- From 01/06/2017 to 01/07/2017 as the ICO Scam decreases the Phishing Scam increases.
- From 01/06/2017 to 01/09/2017 ICO Scam and Phishing Scam have an inverse relationship.
- From 01/05/2017 and 01/07/2018, the Phishing scam peaks and falls off(goes offline). Then from 01/07/2018 to 01/11/2018 the Scamming Scam peaks and fall off(goes offline). So, it can be said that between 01/05/2017 and 01/11/2018 the Phishing Scam and the Scamming Scam have an inverse relationship.
- Over time from the start of the dataset Phishing scam peaked and went offline followed by Scamming Scam peaking and going offline. The ICO Scam did peak during the same time as Phishing Scam but did not fall off(offline) as fast as phishing did.
- ICO Scam may be related to Phishing scam as they peaked at a similar time. However ICO Scam did not peak as high as Phishing Scam, but ICO Scam went offline over a long period of time.

Bibliography

Anon., 2021. *Contract 0xE94b04a0FeD112f3664e45adb2B8915693dD5FF3.* [Online]
Available at: https://etherscan.io/address/0xe94b04a0fed112f3664e45adb2b8915693dd5ff3
[Accessed 02 12 2021].

Bittrex Team, 2021. *Layer 2 Demystified.* [Online]
Available at: https://bittrex.com/discover/layer-2-demystified
[Accessed 02 12 2021].

FRANKENFIELD, J., 2021. *Gas (Ethereum).* [Online]
Available at: https://www.investopedia.com/terms/g/gas-ethereum.asp
[Accessed 02 12 2021].

HumanProgress.org, 2015. *Average cost of RAM per gigabyte.* [Online]
Available at: https://www.humanprogress.org/dataset/average-cost-of-ram-per-gigabyte/
[Accessed 02 12 2021].

Sharma, R., 2019. *What Is Ethereum's "Difficulty Bomb"?.* [Online]
Available at: https://www.investopedia.com/news/what-ethereums-difficulty-bomb/
[Accessed 02 12 2021].