

Part A – Time Analysis (20%)

Create a bar plot showing the number of transactions occurring every month between the start and end of the dataset.

Note: As the dataset spans multiple years and you are aggregating together all transactions in the same month, make sure to include the year in your analysis.

Note: Once the raw results have been processed within Hadoop/Spark you may create your bar plot in any software of your choice (excel, python, R, etc.)

Code: atimeanalysis.py

```
1
2  from mrjob.job import MRJob
3  import time
4  from datetime import datetime
5
6  class PartA(MRJob):
7
8      def mapper(self, _, line):
9          fields = line.split(",")
10
11         try:
12             if (len(fields)==7):
13                 time_epoch = int(fields[6])
14                 month = str(datetime.fromtimestamp(time_epoch).strftime("%B"))
15                 year = str(datetime.fromtimestamp(time_epoch).strftime("%Y"))
16                 date = str(month + " " + year)
17
18                 yield (date, 1)
19
20         except:
21             pass
22
23     def combiner(self, date, count):
24         yield (date, sum(count))
25
26     def reducer(self, date, count):
27         yield (date, sum(count))
28
29 if __name__ == '__main__':
30     PartA.JOBCONF = { 'mapreduce.job.reduces': '3' }
31     PartA.run()
```

Code Explanation:

- Mapper splits the csv data wherever it sees commas (indicating fields)
- Using a try, except statement just in case so it does not capture any malformed data
- Used validation to make sure I have the transactions table in the right format via the if statement
- Converted the timestamp from Unix time and extracted the relevant data (month and year)
- Mapper yields a key value pair of a date (sorted via month) in transactions table with a count of 1
- Combiner and Reducer aggregates these key/value pairs and yields monthly data for how many transactions occurred per month

Code Performance:

Job job_1637317090236_8341 completed successfully

Output directory: hdfs:///user/sa386/output

Counters: 57

File Input Format Counters

Bytes Read=65309532836

File Output Format Counters

Bytes Written=1058

File System Counters

FILE: Number of bytes read=12894

FILE: Number of bytes written=246354442

FILE: Number of large read operations=0

FILE: Number of read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=65309715788

HDFS: Number of bytes read erasure-coded=0

HDFS: Number of bytes written=1058

HDFS: Number of large read operations=0

HDFS: Number of read operations=3282

HDFS: Number of write operations=6

Job Counters

Data-local map tasks=1067

Failed map tasks=14

Launched map tasks=1103

Launched reduce tasks=3

Other local map tasks=14

Rack-local map tasks=22

Total megabyte-milliseconds taken by all map tasks=78650680320

Total megabyte-milliseconds taken by all reduce tasks=3053557760

Total time spent by all map tasks (ms)=15361461

Total time spent by all maps in occupied slots (ms)=76807305

Total time spent by all reduce tasks (ms)=596398

Total time spent by all reduces in occupied slots (ms)=2981990

Total vcore-milliseconds taken by all map tasks=15361461

Total vcore-milliseconds taken by all reduce tasks=596398

Map-Reduce Framework

CPU time spent (ms)=5827000

Combine input records=486521365

Combine output records=1529

Failed Shuffles=0

GC time elapsed (ms)=78869

Input split bytes=182952

Map input records=486522454

Map output bytes=7762895210

Map output materialized bytes=87184

Map output records=486521365

Merged Map outputs=3267

Peak Map Physical memory (bytes)=672645120

Peak Map Virtual memory (bytes)=2971394048

Peak Reduce Physical memory (bytes)=412856320

Peak Reduce Virtual memory (bytes)=2762522624

Physical memory (bytes) snapshot=677343735808

Reduce input groups=47

Reduce input records=1529

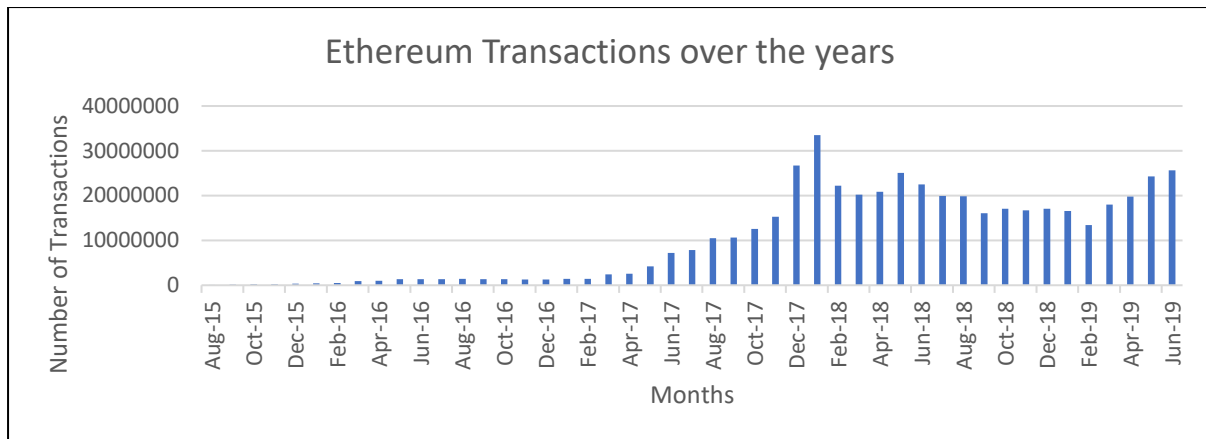
Reduce output records=47

Reduce shuffle bytes=87184

Results/Sample Results:

"April 2017"	2540358
"August 2015"	85112
"August 2018"	19843574
"December 2016"	1316131
"February 2018"	22231978
"January 2016"	404816
"January 2019"	16569597
"July 2016"	1351177
"June 2017"	7240723
"March 2018"	20241701
"May 2017"	4240928
"November 2016"	1301586
"October 2015"	205213
"October 2018"	17070582
"September 2017"	10677965
"April 2018"	20868405
"August 2016"	1411435
"December 2017"	26687692
"February 2016"	520040
"February 2019"	13413899
"January 2017"	1409664
"July 2017"	7839822
"June 2018"	22477336
"March 2016"	916363
"March 2019"	18012971
"May 2018"	25107353
"November 2017"	15292269
"October 2016"	1330960
"September 2015"	174134
"September 2018"	16065654
"April 2016"	1022504
"April 2019"	19821024
"August 2017"	10517032
"December 2015"	347092
"December 2018"	17107601
"February 2017"	1410048
"January 2018"	33504270
"July 2018"	19934164
"June 2016"	1351631
"June 2019"	25646697
"March 2017"	2422460
"May 2016"	1347042
"May 2019"	24325151
"November 2015"	234733
"November 2018"	16713911
"October 2017"	12579172
"September 2016"	1387395

Graph(s):



Graph(s) Explanation:

- Graph shows a negative skew
- The number of Ethereum transactions have increased throughout the years in the provided dataset
- The greatest number of Ethereum transactions occurred during January 2018 with 33504270 transactions
- The fewest number of Ethereum transactions occurred during the initiation of the dataset in the month of August 2015 with 85112 transactions
- From August 2015 to roughly April 2017, the change in number of Ethereum transactions is very minimal but from that point on, the transactions increase by a greater amount
- Reasons as to why Ethereum transactions have increased may be due to cryptocurrency becoming more popular and adopted in today's society

Create a bar plot showing the average value of transaction in each month between the start and end of the dataset.

Code: az2timeanalysis.py

```
1 from mrjob.job import MRJob
2 import time
3 from datetime import datetime
4
5 class PartA2(MRJob):
6
7     def mapper(self, _, line):
8         fields = line.split(",")
9
10
11         try:
12             if (len(fields)==7):
13                 value = int(fields[3])
14
15                 time_epoch = int(fields[6])
16                 month = str(datetime.fromtimestamp(time_epoch).strftime("%B"))
17                 year = str(datetime.fromtimestamp(time_epoch).strftime("%Y"))
18                 date = str(month + " " + year)
19
20
21                 yield (date, (value,1))
22
23         except:
24             pass
25
26     def combiner(self, date, values):
27         wei = 0
28         count = 0
29         for x in values:
30             wei += x[0]
31             count += x[1]
32         yield (date, (wei,count))
33
34     def reducer(self, date, values):
35         wei = 0
36         count = 0
37         for x in values:
38             wei += x[0]
39             count += x[1]
40         yield (date, (wei/count))
41
42 if __name__ == '__main__':
43     PartA2.JOBCONF = { 'mapreduce.job.reduces': '3' }
44     PartA2.run()
```

Code Explanation:

- Mapper splits the csv data wherever it sees commas (indicating fields)
- Using a try, except statement just in case so it does not capture any malformed data
- Used validation to make sure I have the transactions table in the right format via the if statement
- Converted the timestamp from Unix time and extracted the relevant data (month and year)
- Assigned the value of transaction field to the variable called value
- Mapper yields the key-value pair of date-value with the value being its own key-value pair and giving it a count of 1.
- Combiner assigns the value of the transaction to the variable called Wei. It assigns the count that came with it to another variable called count which will increment by 1 each time it receives a new data entry. Wei variable totals up all of the value of the transactions per month.
- Reducer does the exact same as the combiner but the only different thing it does is that it yields the average rather than the aggregation. It does this by dividing the total value of all transactions per month by the count which shows how many entries have been submitted.

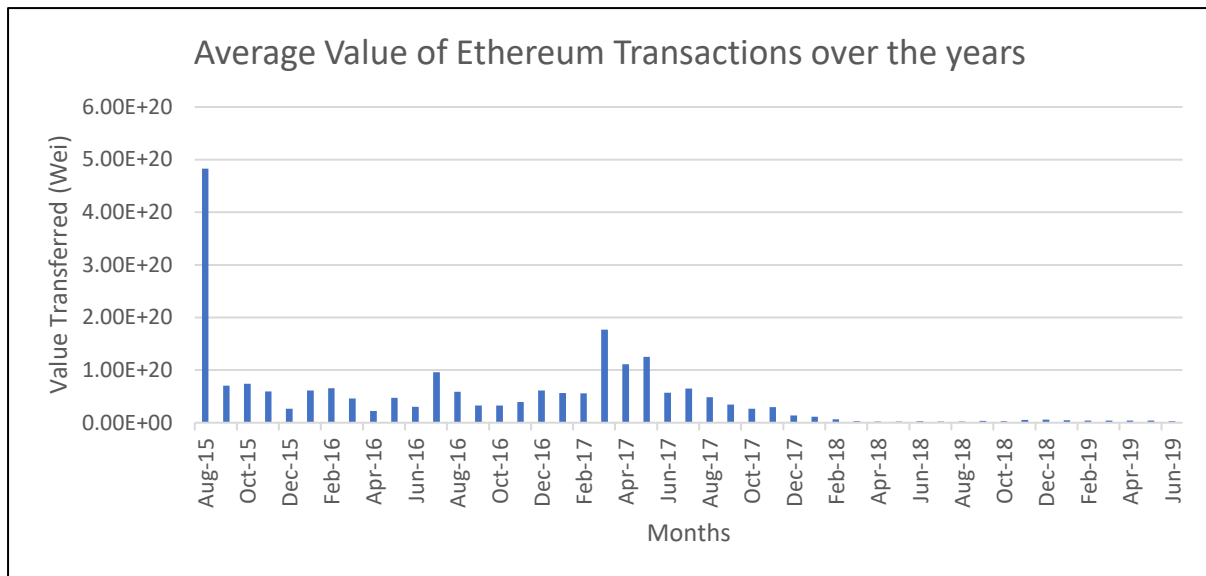
Code Performance:

Output directory: hdfs:///user/sa386/output
Counters: 57
File Input Format Counters
 Bytes Read=65309532836
File Output Format Counters
 Bytes Written=1707
File System Counters
 FILE: Number of bytes read=49134
 FILE: Number of bytes written=246464830
 FILE: Number of large read operations=0
 FILE: Number of read operations=0
 FILE: Number of write operations=0
 HDFS: Number of bytes read=65309715788
 HDFS: Number of bytes read erasure-coded=0
 HDFS: Number of bytes written=1707
 HDFS: Number of large read operations=0
 HDFS: Number of read operations=3282
 HDFS: Number of write operations=6
Job Counters
 Data-local map tasks=1084
 Failed map tasks=5
 Launched map tasks=1094
 Launched reduce tasks=3
 Other local map tasks=5
 Rack-local map tasks=5
 Total megabyte-milliseconds taken by all map tasks=92573271040
 Total megabyte-milliseconds taken by all reduce tasks=3059788800
 Total time spent by all map tasks (ms)=18080717
 Total time spent by all maps in occupied slots (ms)=90403585
 Total time spent by all reduce tasks (ms)=597615
 Total time spent by all reduces in occupied slots (ms)=2988075
 Total vcore-milliseconds taken by all map tasks=18080717
 Total vcore-milliseconds taken by all reduce tasks=597615
Map-Reduce Framework
 CPU time spent (ms)=8867180
 Combine input records=486521365
 Combine output records=1529
 Failed Shuffles=0
 GC time elapsed (ms)=90874
 Input split bytes=182952
 Map input records=486522454
 Map output bytes=14830721503
 Map output materialized bytes=131070
 Map output records=486521365
 Merged Map outputs=3267
 Peak Map Physical memory (bytes)=675266560
 Peak Map Virtual memory (bytes)=2971971584
 Peak Reduce Physical memory (bytes)=406863872
 Peak Reduce Virtual memory (bytes)=2751234048
 Physical memory (bytes) snapshot=683714887680
 Reduce input groups=47

Results/Sample Results:

"April 2017"	1.1102866776414917e+20
"August 2015"	4.830278498461239e+20
"August 2018"	2.3998220839240796e+18
"December 2016"	6.146658677538069e+19
"February 2018"	6.230362795090817e+18
"January 2016"	6.106607047719591e+19
"January 2019"	4.454813888902492e+18
"July 2016"	9.615239675838104e+19
"June 2017"	5.679086208124881e+19
"March 2018"	2.7292981690954767e+18
"May 2017"	1.253013552012346e+20
"November 2016"	3.964431643835122e+19
"October 2015"	7.411289229426749e+19
"October 2018"	3.071710223249729e+18
"September 2017"	3.442063270447764e+19
"April 2018"	2.449338018958015e+18
"August 2016"	5.884337037279917e+19
"December 2017"	1.3737046580115218e+19
"February 2016"	6.55476087599054e+19
"February 2019"	3.980845264511403e+18
"January 2017"	5.620285956535166e+19
"July 2017"	6.4943127187730784e+19
"June 2018"	2.8093034784699515e+18
"March 2016"	4.587940593561187e+19
"March 2019"	3.82001025199019e+18
"May 2018"	2.496999942396033e+18
"November 2017"	2.96411032747405e+19
"October 2016"	3.243501960145934e+19
"September 2015"	7.047276356817186e+19
"September 2018"	3.729156680838141e+18
"April 2016"	2.258293459946353e+19
"April 2019"	4.1346612995334216e+18
"August 2017"	4.827696056983337e+19
"December 2015"	2.6764096183940583e+19
"December 2018"	5.944894163484742e+18
"February 2017"	5.558009016262998e+19
"January 2018"	1.11645727207502e+19
"July 2018"	2.274899868067751e+18
"June 2016"	3.05045289883952e+19
"June 2019"	3.03565822278106e+18
"March 2017"	1.770019614632263e+20
"May 2016"	4.710482015183497e+19
"May 2019"	4.0041423421996964e+18
"November 2015"	5.948474386250283e+19
"November 2018"	5.397909048901716e+18
"October 2017"	2.6731107087869956e+19
"September 2016"	3.2645344559816933e+19

Graph(s):



Graph(s) Explanation:

- The average value of Ethereum transactions over the years provided has a positive skew
- Each Ethereum transaction had a greater amount of Wei transferred on average during the start of the dataset compared to later on in the dataset. This is due to the price of Ethereum in 2015 compared to 2019. Ethereum was much cheaper in 2015 than in 2019 and so each person on the blockchain at that time would tend to have owned more Ethereum on average.
- As Ethereum increased in price, so did the drop off in the average value of Ethereum transactions as shown by the diagram
- The highest average value of Ethereum transactions occurred at the start in August 2015 where the average value transferred in Wei was, 483027849846123000000
- The lowest average value of Ethereum transactions occurred later on in the dataset in July 2018 where the average value transferred in Wei was, 2274899868067750000
- Around Feb 2018, the average value of each transactions has decreased by quite a bit from the start of the dataset due to an influx of people on the blockchain making smaller investments and overall price change of Ethereum possibly

Part B – Top Ten Most Popular Services (20%)

Evaluate the top 10 smart contracts by total Ether received. An outline of the subtasks required to extract this information is provided below, focusing on a MRJob based approach. This is, however, only one possibility, with several other viable ways of completing this assignment.

JOB 1 - INITIAL AGGREGATION

To workout which services are the most popular, you will first have to aggregate **transactions** to see how much each address within the user space has been involved in. You will want to aggregate **value** for addresses in the **to_address** field. This will be similar to the wordcount that we saw in Lab 1 and Lab 2.

Code: bz1topten.py

```
2  from mrjob.job import MRJob
3  import time
4  from datetime import datetime
5
6  class PartB1(MRJob):
7
8      def mapper(self, _, line):
9          fields = line.split(",")
10
11         try:
12             if (len(fields)==7):
13                 toadd = fields[2]
14                 value = int(fields[3])
15
16                 yield (toadd, value)
17         except:
18             pass
19
20     def combiner(self, toadds, values):
21         yield (toadds, sum(values))
22
23     def reducer(self, toadds, values):
24         yield (toadds, sum(values))
25
26 if __name__ == '__main__':
27     PartB1.run()
28
```

Code Explanation:

- Mapper splits the csv data wherever it sees commas (indicating fields)
- Using a try, except statement just in case so it does not capture any malformed data
- Used validation to make sure I have the transactions table in the right format via the if statement
- Put the values of the to_address field in the variable called toadd
- Put the values of the transaction value field in the variable called value
- Yielded the key-value pairs of the address and value fields
- Combiner and Reducer aggregate the values for each specific address reducing any duplicate keys so there are only unique addresses which then gets yielded

Code Performance:

```
Output directory: hdfs:///user/sa386/part1
Counters: 57
File Input Format Counters
  Bytes Read=65309532836
File Output Format Counters
  Bytes Written=3402976383
File System Counters
  FILE: Number of bytes read=4352196753
  FILE: Number of bytes written=12269670611
  FILE: Number of large read operations=0
  FILE: Number of read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=65309715788
  HDFS: Number of bytes read erasure-coded=0
  HDFS: Number of bytes written=3402976383
  HDFS: Number of large read operations=0
  HDFS: Number of read operations=3277
  HDFS: Number of write operations=4
Job Counters
  Data-local map tasks=1076
  Failed map tasks=12
  Launched map tasks=1101
  Launched reduce tasks=2
  Other local map tasks=12
  Rack-local map tasks=13
  Total megabyte-milliseconds taken by all map tasks=81180477440
  Total megabyte-milliseconds taken by all reduce tasks=17920389120
  Total time spent by all map tasks (ms)=15855562
  Total time spent by all maps in occupied slots (ms)=79277810
  Total time spent by all reduce tasks (ms)=3500076
  Total time spent by all reduces in occupied slots (ms)=17500380
  Total vcore-milliseconds taken by all map tasks=15855562
  Total vcore-milliseconds taken by all reduce tasks=3500076
Map-Reduce Framework
  CPU time spent (ms)=10153360
  Combine input records=638961275
  Combine output records=244222762
  Failed Shuffles=0
  GC time elapsed (ms)=103963
  Input split bytes=182952
  Map input records=486522454
  Map output bytes=27398697491
  Map output materialized bytes=7671515959
  Map output records=486521365
  Merged Map outputs=2178
  Peak Map Physical memory (bytes)=670478336
  Peak Map Virtual memory (bytes)=2973360128
  Peak Reduce Physical memory (bytes)=1106833408
```

Results/Sample Results:

[illegible]

JOB 2 - JOINING TRANSACTIONS/CONTRACTS AND FILTERING

Once you have obtained this aggregate of the transactions, the next step is to perform a repartition join between this aggregate and **contracts** (example [here](#)). You will want to join the **to_address** field from the output of Job 1 with the **address** field of **contracts**

Secondly, in the reducer, if the address for a given aggregate from Job 1 was not present within **contracts** this should be filtered out as it is a user address and not a smart contract.

Code: bz2topten.py

```
1 from mrjob.job import MRJob
2 import time
3 from datetime import datetime
4 import re
5
6
7 class PartB2(MRJob):
8
9     def mapper(self, _, line):
10
11         try:
12             #touniqueaddress, aggregation | cut down transactions
13             if len(line.split('\t')) == 2:
14                 fields = line.split('\t')
15                 join_key = fields[0] #to_address
16                 refined_join_key = join_key[1:-1] #to cater for the speech marks
17                 join_value = int(fields[1]) #aggregated_value
18                 yield(refined_join_key, (join_value,1))
19
20             #address, is_erc20, is_erc721, block_number, block_timestamp | contracts
21             if len(line.split(',')) == 5:
22                 fields = line.split(',')
23                 join_key = fields[0] #address
24                 join_value = int(fields[3]) #block_number
25                 yield(join_key, (join_value,2))
26         except:
27             pass
28
29     def reducer(self, address, values):
30         blocknum = 0
31         aggvalue = 0
32
33         for value in values:
34             if value[1]==1:
35                 aggvalue = value[0]
36             elif value[1]==2:
37                 blocknum = value[0]
38             if aggvalue != 0 and blocknum != 0:
39                 yield (address, aggvalue)
40
41
42
43 if __name__ == '__main__':
44     PartB2.JOBCONF = { 'mapreduce.job.reduces': '3' }
45     PartB2.run()
```

Code Explanation:

- Using a try, except statement just in case so it does not capture any malformed data
- For table 1, used validation to make sure I have the right table (output from previous job) in the right format via the if statement with the 2 fields I asked for
- Mapper splits the csv data wherever it sees indentations (indicating fields) because that's how the output file was formatted
- Set the key as the address field and set the value as the aggregated value field. Had to refine the address cos it was displayed within quotation marks in the csv file so needed to extract it.

- Yielded this information with the value section having another key-value pair with the value set to 1. This is used so we could differentiate the tables in the reducer.
- For table 2, used validation to make sure I have the right table, contracts, in the right format via the if statement
- Mapper splits the csv data wherever it sees commas (indicating fields) because that's how the csv file for contracts is formatted
- Set the key as the address field and set the value as some random field. It is random cos we don't specifically need it but using it to check the join works. The value in this was the block number field.
- Yielded this information with the value section having another key-value pair with the value set to 2. This is used so we could differentiate the tables in the reducer.
- Reducer starts off with initializing two container variables called blocknum and aggvalue. For every value that gets sent to the reducer, from each key-value pair, the reducer will set different statements for information retrieved from the different table it received from the mapper.
- The main idea is the same as it adds the value it receives to the container variables mentioned earlier
- If there is a join possible, it will check via the final if statement if both container variables have some data in it at the same time. If it does, it will yield a key-value pair of address of the smart contract and aggregate value of transactions. If not, it will leave out this information as it is not needed.

Code Performance:

```
Output directory: hdfs:///user/sa386/part2
Counters: 57
File Input Format Counters
  Bytes Read=5069456544
File Output Format Counters
  Bytes Written=75052171
File System Counters
  FILE: Number of bytes read=3436784619
  FILE: Number of bytes written=6889923721
  FILE: Number of large read operations=0
  FILE: Number of read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=5069461271
  HDFS: Number of bytes read erasure-coded=0
  HDFS: Number of bytes written=75052171
  HDFS: Number of large read operations=0
  HDFS: Number of read operations=138
  HDFS: Number of write operations=6
Job Counters
  Data-local map tasks=32
  Failed map tasks=2
  Launched map tasks=43
  Launched reduce tasks=3
  Other local map tasks=2
  Rack-local map tasks=9
  Total megabyte-milliseconds taken by all map tasks=6728007680
  Total megabyte-milliseconds taken by all reduce tasks=7019422720
  Total time spent by all map tasks (ms)=1314064
  Total time spent by all maps in occupied slots (ms)=6570320
```

Results/Sample Results:

"0x0000000000b3f879cb30fe243b4dfce438691c04"	210000000000000005
"0x00000000016697fa9a9c8e2889e28d3d9816a078"	4200000000000000000
"0x00000000c4105e2340df06bf6e9358de36a86c7f"	4000100000000000000
"0x000000007b0390fc9ca72f534366f5c02d5af5334"	189583500000000000
"0x000028f957ed4d7b0a7f2b497af7fe44fb8d492a"	7600200000000000000
"0x00005a6d3f88409396844eccef51f87e3c2449f3"	2055020000000000000
"0x00009ccbed893482507888d4b357c13bb8aac4d0"	2951678918000000000
"0x0000b3c60adf7cce979886e09ad7c5c063e8364e"	5608965991561163482
"0x0001027c86c07b4759a0fa93b35ed6e6d9f618ca"	9840000000000000000
"0x000110565a550f03b8aeb2a321228c155e3f3f55"	28007310569200000000
"0x000120d88580557df95c1b0b052e4193d03c436c"	5700000000000000000
"0x000145a905bfca003a3fa4e3ff2fb6070d856566"	9900000000000000000
"0x00015d39f90b9f647aa9d9771dadf9df880e8f8a"	3000000000000000000
"0x000180e8ca31a690ba5b772667938ea9da12d492"	6354200000000000000
"0x00019ad131efb14b3431d9f4f68db5f5922bf1b9"	2894120280000000000
"0x0001a3c6be5a99c2fd89eb00199cf426d20d5acd"	5420000000000000000
"0x0001ad65c82974d70889d74246e33e391e2d7903"	2500000000000000000
"0x0001adf843ffff7fdb5c6791df5bc56cf0ca574f"	2454511982000000000
"0x0001b871ad660424120a9df84529039104248a57"	956233200000000000
"0x0001f643db53ab7f35cc44f796054778b211ff52"	4387739000000000000
"0x0002325fcaaac6ebf1254a626589147bde1a2394"	1
"0x00024105d20b0ead1df31ef9b82edad1beed2e82"	1206543760000000000
"0x00027348648e9b21a803241ad41d7aedee308f11"	8000000000000000000
"0x000274390184ed5986832f9f7fcf9b679b2f12cb"	2677061810000000000
"0x00028c3455500c4458b1a153c536645aad5b6d0"	1153574200000000000
"0x0002d1bd02db58653555c8cca521dab59279b923"	2322838240000000000
"0x0003005b350036b0a1a0761786af6d755dc2720a"	5584006186000000000

JOB 3 - TOP TEN

Finally, the third job will take as input the now filtered address aggregates and sort these via a top ten reducer, utilising what you have learned from lab 4.

Code: bz3topten.py

```
1 |
2 | from mrjob.job import MRJob
3 | import time
4 | from datetime import datetime
5 |
6 | class PartB3(MRJob):
7 |
8 |     def mapper(self, _, line):
9 |         fields = line.split("\t")
10 |
11 |         try:
12 |             #access the fields you want, assuming the format is correct now
13 |             if (len(fields)==2):
14 |                 address = fields[0]
15 |                 aggvalue = int(fields[1])
16 |                 yield(None, (address, aggvalue))
17 |
18 |         except:
19 |             pass
20 |             #no need to do anything, just ignore the line, as it was malformed
21 |
22 |     def combiner(self, _, values):
23 |         sorted_values = sorted(values,reverse=True,key=lambda tup:tup[1])
24 |         i=0
25 |         for value in sorted_values:
26 |             yield("top",value)
27 |             i+=1
28 |             if i >= 10:
29 |                 break
30 |
31 |     def reducer(self, _, values):
32 |         sorted_values = sorted(values,reverse=True,key=lambda tup:tup[1])
33 |         i=0
34 |         for value in sorted_values:
35 |             yield("{} - {}".format(value[0], value[1]), None)
36 |             i+=1
37 |             if i >= 10:
38 |                 break
39 |
40 | if __name__ == '__main__':
41 |     PartB3.JOBCONF = { 'mapreduce.job.reduces': '3' }
42 |     PartB3.run()
43 |
```

Code Explanation:

- Mapper uses a try, except statement just in case so it does not capture any malformed data
- Used validation to make sure I have the right table (output from previous job) in the right format via the if statement with the 2 fields I asked for
- Mapper splits the csv data wherever it sees indentations (indicating fields) because that's how the output file was formatted for the previous section
- Set the contract address from the previous output in a variable called address and set the aggregate ether from the previous output in a variable called aggvalue. The mapper is then tasked with yielding this information but this time we do not assign it a key and pass them both as values in a list format.
- Combiner takes this information and using a lambda function, it sorts the entries by order of the aggvalue from largest to smallest with its associated address.
- It then extracts the top ten from this variable list using a counter and yields this information until it hits 10 entries.
- Reducer takes this information and using a lambda function, it does the same thing as the combiner. The only different thing it does is that it formats the data in a presentable manner until all

10 spaces have been filled which is set via the counter.

Code Performance:

```
Output directory: hdfs:///user/sa386/part3
Counters: 55
File Input Format Counters
  Bytes Read=75052171
File Output Format Counters
  Bytes Written=829
File System Counters
  FILE: Number of bytes read=2173
  FILE: Number of bytes written=1357028
  FILE: Number of large read operations=0
  FILE: Number of read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=75052507
  HDFS: Number of bytes read erasure-coded=0
  HDFS: Number of bytes written=829
  HDFS: Number of large read operations=0
  HDFS: Number of read operations=24
  HDFS: Number of write operations=6
Job Counters
  Data-local map tasks=1
  Launched map tasks=3
  Launched reduce tasks=3
  Rack-local map tasks=2
  Total megabyte-milliseconds taken by all map tasks=181529600
  Total megabyte-milliseconds taken by all reduce tasks=44252160
  Total time spent by all map tasks (ms)=35455
  Total time spent by all maps in occupied slots (ms)=177275
  Total time spent by all reduce tasks (ms)=8643
  Total time spent by all reduces in occupied slots (ms)=43215
  Total vcore-milliseconds taken by all map tasks=35455
  Total vcore-milliseconds taken by all reduce tasks=8643
Map-Reduce Framework
  CPU time spent (ms)=10510
  Combine input records=1160602
  Combine output records=30
  Failed Shuffles=0
  GC time elapsed (ms)=366
  Input split bytes=336
  Map input records=1160602
  Map output bytes=88979395
  Map output materialized bytes=2374
  Map output records=1160602
  Merged Map outputs=9
  Peak Map Physical memory (bytes)=627806208
  Peak Map Virtual memory (bytes)=2736078848
  Peak Reduce Physical memory (bytes)=347148288
```


Results/Sample Results:

This is the result of the top 10 smart contracts based on the amount of ether it receives. It is separated by the address of the contract and all of the ether it has received.

```
1 "\"0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444\" - 84155100809965865822726776" null
2 "\"0xfa52274dd61e1643d2205169732f29114bc240b3\" - 45787484483189352986478805" null
3 "\"0x7727e5113d1d161373623e5f49fd568b4f543a9e\" - 45620624001350712557268573" null
4 "\"0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef\" - 43170356092262468919298969" null
5 "\"0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8\" - 27068921582019542499882877" null
6 "\"0xbfc39b6f805a9e40e77291aff27aee3c96915bdd\" - 21104195138093660050000000" null
7 "\"0xe94b04a0fed112f3664e45adb2b8915693dd5ff3\" - 15562398956802112254719409" null
8 "\"0xbb9bc244d798123fde783fcc1c72d3bb8c189413\" - 11983608729202893846818681" null
9 "\"0xabbb6bebf05aa13e908eaa492bd7a8343760477\" - 11706457177940895521770404" null
10 "\"0x341e790174e3a4d35b65fdc067b6b5634a61caea\" - 8379000751917755624057500" null
11
```

Part C – Top Ten Most Active Miners (10%)

Evaluate the top 10 miners by the size of the blocks mined. This is simpler as it does not require a join. You will first have to aggregate **blocks** to see how much each miner has been involved in. You will want to aggregate **size** for addresses in the **miner** field. This will be similar to the wordcount that we saw in Lab 1 and Lab 2. You can add each value from the reducer to a list and then sort the list to obtain the most active miners.

Part C is split into two sections. One section is dedicated to the aggregation whilst the other is used to refine the first section until we get a top 10.

Code: c1topten.py

```
2  from mrjob.job import MRJob
3  import time
4  from datetime import datetime
5
6  class PartC1(MRJob):
7
8      def mapper(self, _, line):
9          fields = line.split(",")
10
11         try:
12             if (len(fields)==9):
13                 miner = fields[2]
14                 size = int(fields[4])
15
16                 yield (miner, size)
17         except:
18             pass
19
20     def combiner(self, miner, values):
21         yield (miner, sum(values))
22
23     def reducer(self, miner, values):
24         yield (miner, sum(values))
25
26 if __name__ == '__main__':
27     PartC1.JOBCONF = { 'mapreduce.job.reduces': '3' }
28     PartC1.run()
```

Code Explanation:

- Mapper uses a try, except statement just in case so it does not capture any malformed data
- Mapper splits the csv data wherever it sees commas (indicating fields) because that's how the csv file is formatted for the blocks table
- Used validation to make sure I have the right table in the right format via the if statement with all 9 fields it possesses.
- Created a variable called miner which holds the miner address from the blocks table and another variable called size which holds the size of the blocks. This information then gets yielded with the key set as the miner address.
- The combiner and reducer get every single instance of key-value pairs of the same key and add up

all the sizes so there is one key which holds all of the aggregation of the number of blocks mined. This information gets yielded and there should now only be unique miner addresses within the output.

Code Performance:

```
Output directory: hdfs:///user/sa386/part1
Counters: 55
File Input Format Counters
  Bytes Read=1348852922
File Output Format Counters
  Bytes Written=244145
File System Counters
  FILE: Number of bytes read=276463
  FILE: Number of bytes written=8868312
  FILE: Number of large read operations=0
  FILE: Number of read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=1348858268
  HDFS: Number of bytes read erasure-coded=0
  HDFS: Number of bytes written=244145
  HDFS: Number of large read operations=0
  HDFS: Number of read operations=114
  HDFS: Number of write operations=6
Job Counters
  Data-local map tasks=32
  Launched map tasks=33
  Launched reduce tasks=3
  Rack-local map tasks=1
  Total megabyte-milliseconds taken by all map tasks=1274219520
  Total megabyte-milliseconds taken by all reduce tasks=40422400
  Total time spent by all map tasks (ms)=248871
  Total time spent by all maps in occupied slots (ms)=1244355
  Total time spent by all reduce tasks (ms)=7895
  Total time spent by all reduces in occupied slots (ms)=39475
  Total vcore-milliseconds taken by all map tasks=248871
  Total vcore-milliseconds taken by all reduce tasks=7895
Map-Reduce Framework
  CPU time spent (ms)=103370
  Combine input records=8060600
  Combine output records=9727
  Failed Shuffles=0
  GC time elapsed (ms)=2282
  Input split bytes=5346
  Map input records=8060633
  Map output bytes=404039587
  Map output materialized bytes=475466
  Map output records=8060600
  Merged Map outputs=99
  Peak Map Physical memory (bytes)=642711552
  Peak Map Virtual memory (bytes)=2738536448
```

Results/Sample Results:

"0x000083ceb2317f5755be7a745e3c4be7ba396877"	2362
"0x000354c8e89af5a4ea0be126176a14848678a391"	2197
"0x000b800ab6000bc8e5c02423723bbb168720530a"	311949
"0x001147dcbblec632af78203fdfa4981d65f2cd46"	768
"0x0017392862fc742e3436bfb6b4593188227aead7"	4520
"0x00192fb10df37c9fb26829eb2cc623cd1bf599e8"	166469030
"0x0025571697c62d4fe90b06dda1ceae7c6f31df0d"	75440
"0x0025bc2515c073784222cbcd1e988dba9a0f75af"	1748
"0x0027bcf5801c9d21801f8089a926643da06eeb3a"	1316
"0x002a0f8b3d5d866e3ceafb904e7f48741c43026b"	994
"0x0030f2388b2cb4931df260ae05de35e67bf11bbd"	12577
"0x00333fbf07523267d356c1b9b98011e24bea32d5"	37059
"0x003a6cefcd4377033674d8a80ddbfb3596bd13dd"	1545
"0x003fe8a3332f5529dff5bba8e5ba659cdcae735d"	547
"0x0042efe2480f1822d62941ed8d85ede87fa800b8"	7516
"0x00438a1c5a027f2effe7e98ecaa3fe760a1bda9e"	3462
"0x0049d2c3466608a0da924269bcd4afa28bad9fde"	662
"0x004a353163f5b70235409be653d87326df1ec9cc"	4140
"0x0051f312d9ee75f82fa4cb7db3afe45825727085"	525945
"0x0060415d52734811c7d4424377124aeee30cd157"	544
"0x006c505827a645f34ee29212afe569a0ba2d388c"	412169
"0x007286a33536c91736bd10a5d5dfbc742b71ee15"	4384

Code: c2topten.py

```
1 from mrjob.job import MRJob
2 import time
3 from datetime import datetime
4
5 class PartC2(MRJob):
6
7     def mapper(self, _, line):
8         fields = line.split("\t")
9
10        try:
11            #access the fields you want, assuming the format is correct now
12            if (len(fields)==2):
13                address = fields[0]
14                aggvalue = int(fields[1])
15                yield(None, (address, aggvalue))
16
17        except:
18            pass
19            #no need to do anything, just ignore the line, as it was malformed
20
21    def combiner(self, _, values):
22        sorted_values = sorted(values,reverse=True,key=lambda tup:tup[1])
23        i=0
24        for value in sorted_values:
25            yield("top",value)
26            i+=1
27            if i >= 10:
28                break
29
30    def reducer(self, _, values):
31        sorted_values = sorted(values,reverse=True,key=lambda tup:tup[1])
32        i=0
33        for value in sorted_values:
34            yield("{} - {}".format(value[0], value[1]), None)
35            i+=1
36            if i >= 10:
37                break
38
39 if __name__ == '__main__':
40     PartC2.JOBCONF = { 'mapreduce.job.reduces': '3' }
41     PartC2.run()
```

Code Explanation:

- Mapper uses a try, except statement just in case so it does not capture any malformed data
- Mapper splits the csv data wherever it sees indentation (indicating fields) because that's how the output file is formatted for the previous table
- Used validation to make sure I have the right table in the right format via the if statement which makes sure that it has two fields.
- Miner address gets allocated to the address variable and the aggregate of the sizes of the block is stored in the aggvalue variable. These both get yielded within the value section and there is no key.
- Combiner takes this information and using a lambda function, it sorts the entries by order of the aggvalue from largest to smallest with its associated address.
- It then extracts the top ten from this variable list using a counter and yields this information until it hits 10 entries.
- Reducer takes this information and using a lambda function, it does the same thing as the combiner. The only different thing it does is that it formats the data in a presentable manner until all 10 spaces have been filled which is set via the counter.

Code Performance:

```
Output directory: hdfs:///user/sa386/part2
Counters: 54
File Input Format Counters
  Bytes Read=244145
File Output Format Counters
  Bytes Written=673
File System Counters
  FILE: Number of bytes read=1749
  FILE: Number of bytes written=1356278
  FILE: Number of large read operations=0
  FILE: Number of read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=244481
  HDFS: Number of bytes read erasure-coded=0
  HDFS: Number of bytes written=673
  HDFS: Number of large read operations=0
  HDFS: Number of read operations=24
  HDFS: Number of write operations=6
Job Counters
  Launched map tasks=3
  Launched reduce tasks=3
  Rack-local map tasks=3
  Total megabyte-milliseconds taken by all map tasks=55598080
  Total megabyte-milliseconds taken by all reduce tasks=63534080
  Total time spent by all map tasks (ms)=10859
  Total time spent by all maps in occupied slots (ms)=54295
  Total time spent by all reduce tasks (ms)=12409
  Total time spent by all reduces in occupied slots (ms)=62045
  Total vcore-milliseconds taken by all map tasks=10859
  Total vcore-milliseconds taken by all reduce tasks=12409
```

Results/Sample Results:

This is the result of the top 10 miners based on the number of blocks mined. It is separated by the address of the miner and all of the blocks it has mined which is an aggregation of all the block sizes in bytes.

```
1  |\"0xea674fdde714fd979de3edf0f56aa9716b898ec8\" - 23989401188" null
2  |\"0x829bd824b016326a401d083b33d092293333a830\" - 15010222714" null
3  |\"0x5a0b54d5dc17e0aad3c383d2db43b0a0d3e029c4c\" - 13978859941" null
4  |\"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5\" - 10998145387" null
5  |\"0xb2930b35844a230f00e51431acae96fe543a0347\" - 7842595276" null
6  |\"0x2a65aca4d5fc5b5c859090a6c34d164135398226\" - 3628875680" null
7  |\"0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01\" - 1221833144" null
8  |\"0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb\" - 1152472379" null
9  |\"0x1e9939daaad6924ad004c2560e90804164900341\" - 1080301927" null
10 |\"0x61c808d82a3ac53231750dad3c777b59310bd9\" - 692942577" null
11
```

Part D – Data Exploration (50%)

Identifying Contract Types The identification and classification of smart contracts can help us to better understand the behaviour of smart contracts and figure out vulnerabilities, such as confirming fraud contracts. By identifying features of different contracts such as number of transactions, number of unique outflow addresses, Ether balance and others we can identify different types of contracts.

-How many different types can you identify?

-What is the most popular type of contract?

More information can be found at

<https://www.sciencedirect.com/science/article/pii/S0306457320309547#bib0019>. Please note that you are not required to use the types defined in this paper. Partial marks will be awarded for feature extraction and analysis. (25/50)

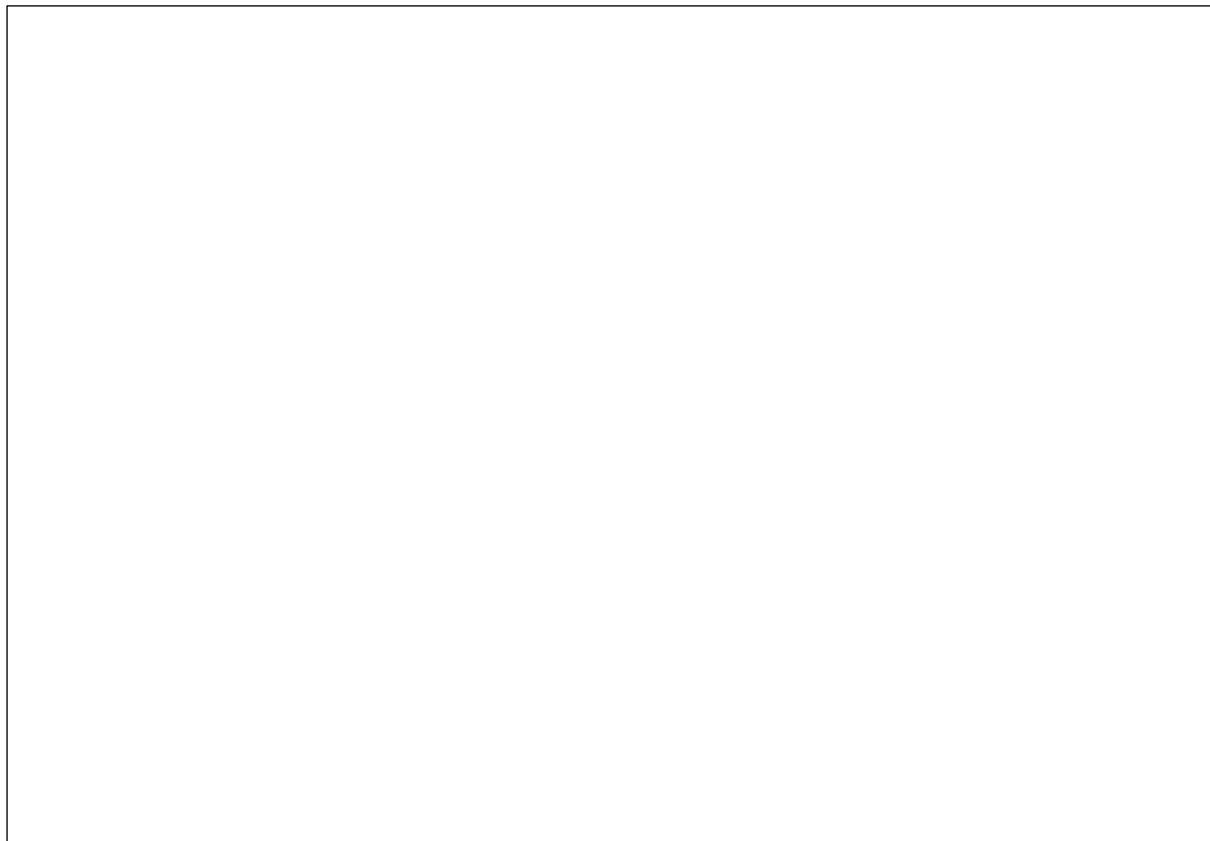
Code:

ab

Code Explanation:

ab

Code Performance:



Results/Sample Results:



Graph(s):

ab

Graph(s) Explanation:

Ab

DIDNT FINISH

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)

Code: jParseEasier.py

```
1 #pip install pandas
2
3 import pandas as pd
4
5 df = pd.read_json("scams.json")
6 df.to_csv("scamseasier.csv")
7
```

Code Explanation:

-To be able to carry out big data tasks, I needed to convert the scams json file to a csv file and parse it

-There is a library called pandas which carries out this task for us and the output can be seen below. Majority of the headers are enclosed in double quotation marks, followed by being inside of a dictionary. So, in order to extract values, we would have to make use of this dictionary.

Results/Sample Results:

```
||,success,result
0x0020731604c882cf7bf8c444be97d17b19ea4316,True,"{'id': 6606, 'name': 'decentralized-
launch.com', 'url': 'http://decentralized-launch.com', 'coin': 'ETH', 'category':
'Scamming', 'subcategory': 'Trust-Trading', 'description': 'Trust trading scam site - fake
BNB giveaway', 'addresses': ['0x0020731604c882cf7bf8c444be97d17b19ea4316',
'16fxzp7wr9xvryrnzgpddbbajexqvj2tdz'], 'reporter': 'MyCrypto', 'status': 'Offline'}"
0x002bf459dc58584d58886169ea0e80f3ca95ffaf,True,"{'id': 3874, 'name': 'ethergifting.com',
'url': 'https://ethergifting.com', 'coin': 'ETH', 'category': 'Scamming', 'subcategory':
'Trust-Trading', 'description': 'Trust trading scam site', 'addresses':
['0x002bf459dc58584d58886169ea0e80f3ca95ffaf'], 'reporter': 'MyCrypto', 'status':
'Offline'}"
0x002f0c8119c16d310342d869ca8bf6ace34d9c39,True,"{'id': 4384, 'name': 'eth-pay.org', 'url':
'http://eth-pay.org', 'coin': 'ETH', 'category': 'Scamming', 'subcategory': 'Trust-
Trading', 'description': 'Trust trading scam site', 'addresses':
['0x002f0c8119c16d310342d869ca8bf6ace34d9c39'], 'reporter': 'MyCrypto', 'status':
'Offline'}"
0x0059b14e35dab1b4ee1e2926c7a5660da66f747,True,"{'id': 2736, 'name': 'iost.co', 'url':
'http://iost.co', 'coin': 'ETH', 'category': 'Phishing', 'subcategory': 'Iost',
'description': 'Fake Isot airdrop site, phishing for private keys', 'addresses':
['0x545f5e5c54d8ac72af1c7de8c070387b73841a24',
'0x0059b14e35dab1b4ee1e2926c7a5660da66f747'], 'reporter': 'MyCrypto', 'ip':
'216.83.57.94', 'nameservers': ['dns16.hichina.com', 'dns15.hichina.com'], 'status':
'Active'}"
0x005b9f4516f8e640bbe48136901738b323c53b00,True,"{'id': 3559, 'name': 'coinbase-eth-
giveaway.com', 'url': 'http://coinbase-eth-giveaway.com', 'coin': 'ETH', 'category':
'Scamming', 'subcategory': 'Trust-Trading', 'description': 'Trust trading scam site',
'addresses': ['0x005b9f4516f8e640bbe48136901738b323c53b00'], 'reporter': 'MyCrypto',
'status': 'Offline'}"
```

DIDNT FINISH

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)

Code: d3gasguzz.py

```
2 from mrjob.job import MRJob
3 import time
4 from datetime import datetime
5
6 class PartD3(MRJob):
7
8     def mapper(self, _, line):
9         fields = line.split(",")
10
11         try:
12             if (len(fields)==7):
13                 gasprice = int(fields[5])
14
15                 time_epoch = int(fields[6])
16                 month = str(datetime.fromtimestamp(time_epoch).strftime("%B"))
17                 year = str(datetime.fromtimestamp(time_epoch).strftime("%Y"))
18                 date = str(month + " " + year)
19
20                 yield (date, (gasprice,1))
21
22         except:
23             pass
24
25     def combiner(self, date, values):
26         price = 0
27         count = 0
28         for x in values:
29             price += x[0]
30             count += x[1]
31         yield (date, (price,count))
32
33     def reducer(self, date, values):
34         price = 0
35         count = 0
36         for x in values:
37             price += x[0]
38             count += x[1]
39         yield (date, (price/count))
40
41
42 if __name__ == '__main__':
43     PartD3.JOBCONF = { 'mapreduce.job.reduces': '3' }
44     PartD3.run()
```

Code Explanation:

- Mapper splits the csv data wherever it sees commas (indicating fields)
- Using a try, except statement just in case so it does not capture any malformed data
- Used validation to make sure I have the transactions table in the right format via the if statement
- Set the gasprice variable to hold the information about the gas_price column in the transactions table and used the date code, as used earlier, to be the key in my program. I then yielded this information, but I also added a count (1) in the values section for the gasprice as I will be calculating an average.
- Combiner assigns the gasprice of the transaction to the variable called price. It assigns the count that came with it to another variable called count which will increment by 1 each time it receives a new data entry. Price variable totals up all of the gasprice per month.
- Reducer does the exact same as the combiner but the only different thing it does is that it yields the average rather than the aggregation. It does this by dividing the total gas price per month by the count which shows how many entries have been submitted and this outputs an average.

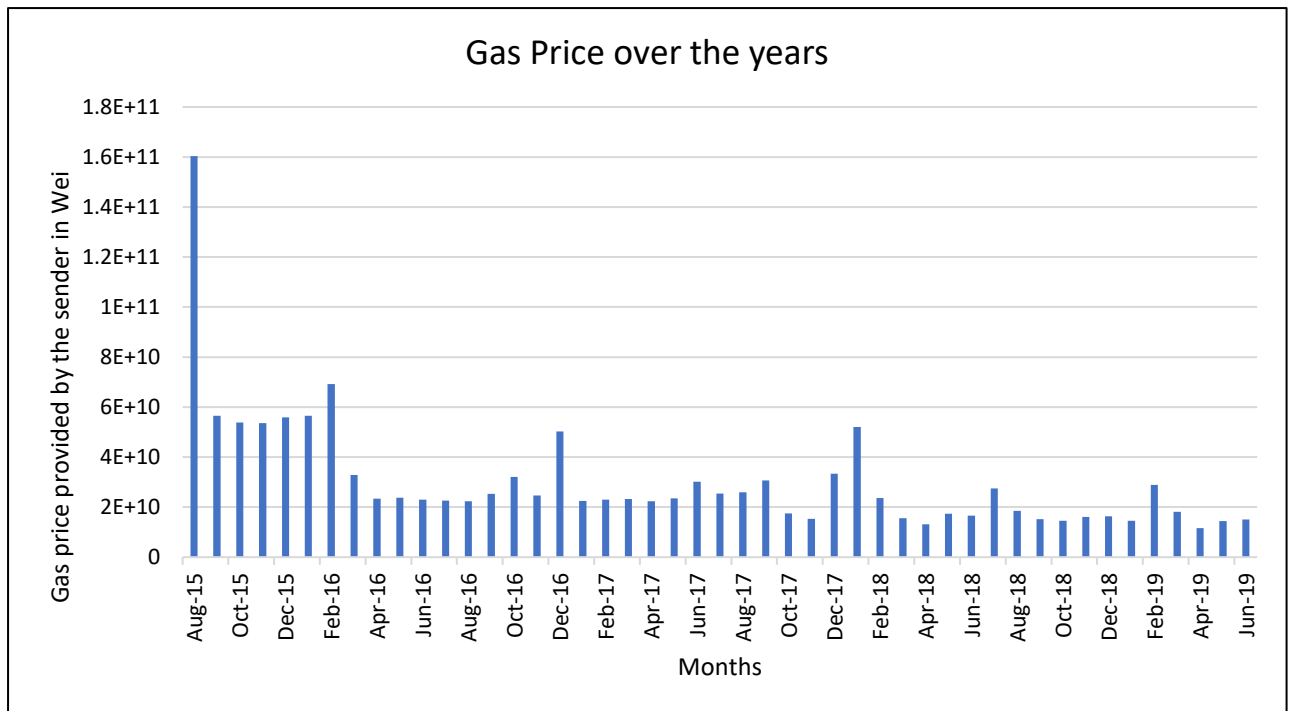
Code Performance:

```
Output directory: hdfs:///user/sa386/output3
Counters: 57
File Input Format Counters
  Bytes Read=65309532836
File Output Format Counters
  Bytes Written=1547
File System Counters
  FILE: Number of bytes read=37506
  FILE: Number of bytes written=246389936
  FILE: Number of large read operations=0
  FILE: Number of read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=65309715788
  HDFS: Number of bytes read erasure-coded=0
  HDFS: Number of bytes written=1547
  HDFS: Number of large read operations=0
  HDFS: Number of read operations=3282
  HDFS: Number of write operations=6
Job Counters
  Data-local map tasks=1085
  Failed map tasks=4
  Launched map tasks=1093
  Launched reduce tasks=3
  Other local map tasks=4
  Rack-local map tasks=4
  Total megabyte-milliseconds taken by all map tasks=94018135040
  Total megabyte-milliseconds taken by all reduce tasks=3045939200
  Total time spent by all map tasks (ms)=18362917
  Total time spent by all maps in occupied slots (ms)=91814585
  Total time spent by all reduce tasks (ms)=594910
  Total time spent by all reduces in occupied slots (ms)=2974550
  Total vcore-milliseconds taken by all map tasks=18362917
  Total vcore-milliseconds taken by all reduce tasks=594910
Map-Reduce Framework
  CPU time spent (ms)=8978890
  Combine input records=486521365
  Combine output records=1529
  Failed Shuffles=0
  GC time elapsed (ms)=110733
  Input split bytes=182952
  Map input records=486522454
  Map output bytes=14852378331
  Map output materialized bytes=117793
  Map output records=486521365
  Merged Map outputs=3267
  Peak Map Physical memory (bytes)=670334976
  Peak Map Virtual memory (bytes)=2972839936
  Peak Reduce Physical memory (bytes)=414355456
  Peak Reduce Virtual memory (bytes)=2747699200
```

Results/Sample Results:

"April 2017"	22357075153.737774
"August 2015"	160356354969.0582
"August 2018"	18478650928.73732
"December 2016"	50318068074.68128
"February 2018"	23636574203.828976
"January 2016"	56596270931.31685
"January 2019"	14611816445.785303
"July 2016"	22619213302.82447
"June 2017"	30201664896.648643
"March 2018"	15554999714.874083
"May 2017"	23568661138.035034
"November 2016"	24634294365.279953
"October 2015"	53898497955.07804
"October 2018"	14527572489.594873
"September 2017"	30676555381.728302
"April 2018"	13149523170.458775
"August 2016"	22407628763.365383
"December 2017"	33423472930.410748
"February 2016"	69180681134.38849
"February 2019"	28940599438.14876
"January 2017"	22507570807.719795
"July 2017"	25463022668.144382
"June 2018"	16536952425.540327
"March 2016"	32805967466.947655
"March 2019"	18091340267.246918
"May 2018"	17414613148.436962
"November 2017"	15312465314.693544
"October 2016"	32113146198.902683
"September 2015"	56512934320.02001
"September 2018"	15208159827.250362
"April 2016"	23359978331.67677
"April 2019"	11573133401.384796
"August 2017"	25903650367.898697
"December 2015"	55899526672.35486
"December 2018"	16338844844.014668
"February 2017"	23047230327.254303
"January 2018"	52106060636.84502
"July 2018"	27520561081.02132
"June 2016"	23021831286.57444
"June 2019"	15067557451.333876
"March 2017"	23232083087.909393
"May 2016"	23747073761.791187
"May 2019"	14480574461.419483
"November 2015"	53607614201.796776
"November 2018"	16034859008.681648
"October 2017"	17509171844.770706
"September 2016"	25262249340.116425

Graph(s):



Graph(s) Explanation:

- The gas price over the years has seen a gradual decrease from the start of the dataset until the very end.
- The highest gas price which was provided by the sender was 160356354969.058 Wei and this occurred in August 2015.
- The lowest gas price which was provided by the sender was 11573133401.3847Wei and this occurred in April 2019.
- The graph has a positive skew.
- Possible reasons as to why gas prices have decreased over the years is that Ethereum price has gone up and as this has occurred mining has become more easier due to increased computational power and advances in technology.

Have contracts become more complicated, requiring more gas, or less so?

Code: d321gasguzz.py

```
2 from mrjob.job import MRJob
3 import time
4 from datetime import datetime
5
6 class PartD3(MRJob):
7
8     def mapper(self, _, line):
9
10         try:
11             fields = line.split(",")
12             if (len(fields)==7):
13                 toadd = fields[2] #toaddress from transactions
14                 agggas = int(fields[4]) #aggregate of gas
15
16                 if agggas == 0:
17                     pass
18                 else:
19                     yield (toadd, agggas)
20         except:
21             pass
22
23     def combiner(self, toadds, values):
24         yield (toadds, sum(values))
25
26     def reducer(self, toadds, values):
27         yield (toadds, sum(values)) #unique toaddress, gas aggregate
28
29 if __name__ == '__main__':
30     PartD3.JOBCONF = { 'mapreduce.job.reduces': '3' }
31     PartD3.run()
32
```

Code Explanation:

- Mapper splits the csv data wherever it sees commas (indicating fields)
- Using a try, except statement just in case so it does not capture any malformed data
- Used validation to make sure I have the transactions table in the right format via the if statement
- Set the toadd variable to hold the information about the to_address column in the transactions table and I set the agggas variable to hold the information about the gas provided by the sender. I also put an if-statement to not even try to process transactions where the gas used was 0 but if it was greater than it, it should and yield the result in a key-value format where the key is the address, and the aggregate is the value.
- Combiner and Reducer aggregates these key/value pairs and yields an aggregate of the gas per unique address.

Results/Sample Results:

[illegible]

Code: d322gasguzz.py

```

2 from mrjob.job import MRJob
3 import time
4 from datetime import datetime
5 import re
6
7 class PartD3(MRJob):
8
9     def mapper(self, _, line):
10
11         try:
12             #touniqueaddress, aggregation | cut down transactions
13             if len(line.split('\t')) == 2:
14                 fields = line.split('\t')
15                 join_key = fields[0] #to_address
16                 refined_join_key = join_key[1:-1] #to cater for the speech marks
17                 join_value = int(fields[1]) #aggregated_value
18                 yield(refined_join_key, (join_value,1))
19
20             #address, is_erc20, is_erc721, block_number, block_timestamp | contracts
21             if len(line.split(',')) == 5:
22                 fields = line.split(',')
23                 join_key = fields[0] #address
24                 join_value = str(fields[4]) #block_timestamp
25                 yield(join_key, (join_value,2))
26
27         except:
28             pass
29
30     def reducer(self, address, values):
31         blocktime = ""
32         aggvalue = 0
33
34         for value in values:
35             if value[1]==1:
36                 aggvalue = value[0]
37             elif value[1]==2:
38                 blocktime = str(value[0])
39
40         if aggvalue != 0 and blocktime != "":
41             yield (address, (aggvalue, blocktime))
42
43 if __name__ == '__main__':
44     PartD3.JOBCONF = { 'mapreduce.job.reduces': '5' }
45     PartD3.run()

```

Code Explanation:

-In this section of the code, we are trying to join the transaction table with contracts table and the common key for this was the address. So, from the transaction table, I deciphered which ones that were listed are contracts. The fields I carried over as the value were the aggregated gas, and timestamp the key was the contract address.

-Add more...

Results/Sample Results:

The format of this is contract, [total gas, timestamp]

```
"0x0000000000001b84b1cb32787b0d64758d019317" [26424804, "2019-06-04 20:20:30 UTC"]
"0x00000000000075efbee23fe2de1bd0b7690883cc9" [392278352, "2018-12-18 02:26:26 UTC"]
"0x000000000000fe8503db73c68f1a1874eb9d86883" [144172, "2019-05-20 03:36:13 UTC"]
"0x00000000000035a5b6bbb4618338cb65f9b3dc1600" [42000, "2019-02-28 19:59:54 UTC"]
"0x00000000000073b6eeb97a1b007fa833561b10be1e" [200000, "2019-02-28 21:06:20 UTC"]
"0x000000000000a3c7c86345a35a0e97a4bb4370a8dd9" [287599222, "2019-04-23 22:22:15 UTC"]
"0x000000000000a8f806c754549943b6550a2594c9a126" [25466203000, "2018-11-17 09:35:14 UTC"]
"0x000000000000b159899f70a67c323e43057fe1b7358f" [4864168, "2019-04-19 09:22:40 UTC"]
"0x000000000000c4105e2340df06bf6e9358de36a86c7f" [82353418, "2019-04-20 10:12:53 UTC"]
"0x000000000000c47cae4dfcbff7afbfae810dc1e99f98" [7792506000, "2018-11-13 00:36:16 UTC"]
"0x000000000000e2b22a1d0dbf1096fa50f252b8325e18a" [7763601184, "2018-11-03 21:18:46 UTC"]
"0x000000000000e4fec2ab4fa3ec2af763248c17973612" [4494480000, "2018-11-09 09:57:50 UTC"]
"0x0000003ed2eb44cded8ade31c01dda60da466b2d1" [233803, "2018-09-18 12:42:56 UTC"]
"0x00001f90ccf3e452d30914471b694d5c4e3d90d6" [450000, "2018-01-01 10:10:55 UTC"]
"0x000031294143ce31a1a772c97fe7f775bf88d1dc" [516816, "2017-12-21 01:35:34 UTC"]
"0x0000444195bd43dd584c7a72cae24dfac0925b9e" [159601, "2018-02-11 13:17:05 UTC"]
"0x0000946e296619df98dc5562d70874a14813b111" [10340000, "2017-10-04 08:47:19 UTC"]
"0x0000c48b539e783537923da2b635ef202a9ffc2a" [50000, "2019-03-21 05:27:16 UTC"]
"0x0000de3c4a8daf93dbe5c4b28aeb486236b82da8" [25289389, "2017-09-03 05:59:05 UTC"]
"0x000117cd6305b2b4d8d5556bcb66365db92043a7" [100000, "2019-05-14 05:17:14 UTC"]
"0x00012d92a0e7ee1b19f8e018267c97a3a7e99aa7" [270000, "2017-05-01 16:54:50 UTC"]
"0x00017df88c326d4bafbf0407c50210c0c7dd4bce9" [94011300, "2018-10-10 07:32:55 UTC"]
```

Code: d323gasguzz.py

```
2 from mrjob.job import MRJob
3 import time
4 from datetime import datetime
5
6 class PartD3(MRJob):
7
8     def mapper(self, _, line):
9         fields = line.split("\t")
10
11         try:
12             #access the fields you want, assuming the format is correct now
13             if (len(fields)==2):
14                 address = fields[0]
15                 aggvalue = int(fields[1][0])
16                 timestamp = str(fields[1][1][1:-1])
17                 yield(None, (address, aggvalue, timestamp))
18
19         except:
20             pass
21             #no need to do anything, just ignore the line, as it was malformed
22
23     def combiner(self, _, values):
24         sorted_values = sorted(values,reverse=True,key=lambda tup:tup[2])
25
26         for value in sorted_values:
27             yield("top",value)
28
29     def reducer(self, _, values):
30         sorted_values = sorted(values,reverse=True,key=lambda tup:tup[2])
31
32         time_epoch = int(value[2])
33         month = str(datetime.fromtimestamp(time_epoch).strftime("%B"))
34         year = str(datetime.fromtimestamp(time_epoch).strftime("%Y"))
35         date = str(month + " " + year)
36
37         for value in sorted_values:
38             yield("{} - {} - {}".format(value[0], value[1], date), None)
39
40
41 if __name__ == '__main__':
42     PartD3.JOBCONF = { 'mapreduce.job.reduces': '10' }
43     PartD3.run()
44
```

Code Explanation:

-In this code, the aim was to use the output from the previous section and reverse order them in timestamp order. I would've then plotted this data onto a graph in excel with the y-axis holding information about the aggregate gas value and the timestamp on the x-axis to see how gas usage has differed over time.

-Add more... (couldn't run due to issues with cluster not working and throwing up different errors)

Results/Sample Results:

DIDNT FINISH

