KUSHAGRA GUPTA-190649351

Coursework

Part A. Time Analysis:

For this task we have to aggregate the transactions occurring every month between the start and end of the dataset. Below is the code that has been used for achieving the results.

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

class course(MRJob):

    def mapper(self, _, line):
        #the field has been splitted with the ','
        fields=line.split(",")
        try:
            if (len(fields)==7):
                #Date has been coverted to string and the epoch time is converted to normal date
                date = int(fields[6])
                #date has been extracted , taking years and months
                year = time.strftime('%Y-%B', time.gmtime(date))
                yield (year,1)
        except:
            pass
#reducer to sum the number of transaction per month
    def reducer(self, year, counts):
        yield (year,sum(counts))

    def combiner(self, year, counts):
        yield (year,sum(counts))


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

Job ID : http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1575381276332_0253/

Code Analysis :

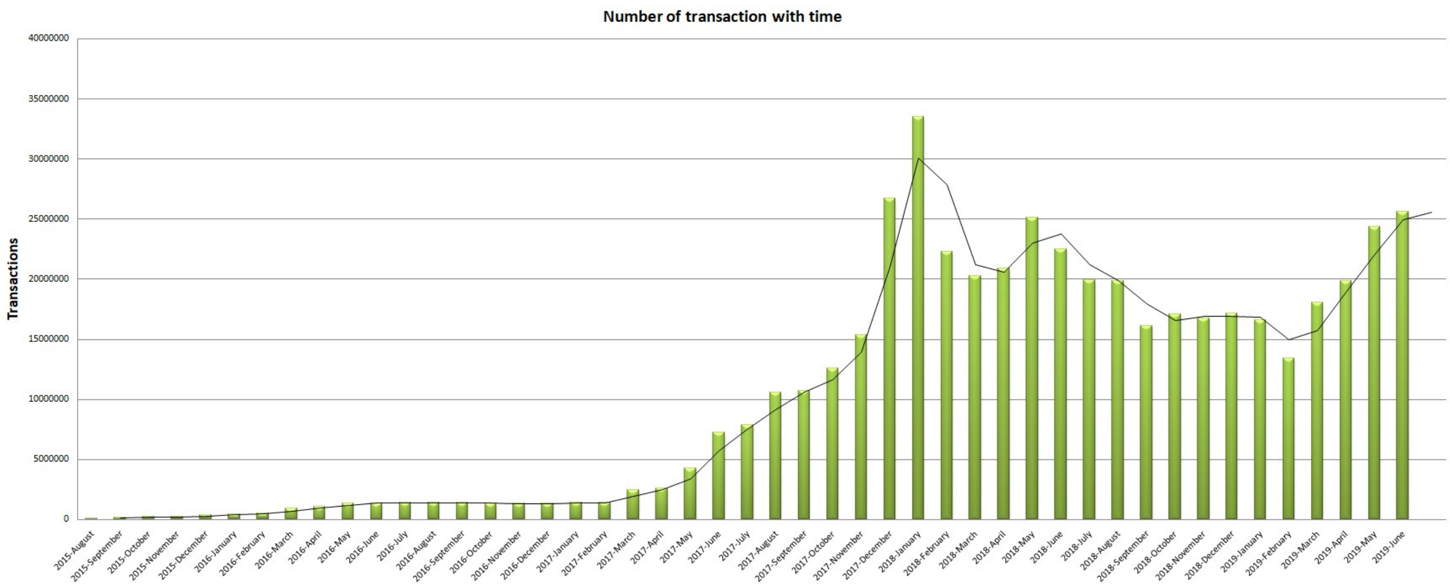
Mapper -

The data has been fetched from transactions dataset. The dataset has 7 fields and all the fields are separated by ‘,’, a try and catch has been used to filter out the values which are correct and pass the code if it doesn’t match the criteria. The date has been taken and converted from epoch time to gmt time and later into month and year. The date has been made the key and has been yield with value of 1 with all maps according to that particular month.

Reducer :

The key and value pair is than reduced and summed according to the months. The result is yielded and we get the aggregation according to the months.

The below graph has been plotted with the result dataset from the above code. The graph has been plotted with ‘Transactions’ on Y-axis and ‘Date’ on X-axis.



From the graph it can be analyzed that there were slight increase in the number of transaction from 2015-August to 2017-May, but there was a surge in transaction from 2017-June to 2018-January and it decreased further going over the years. The transactions attained its high in the month of 2018-January. The Transactions can be seen further increasing at the end of last three months in the graph.

Part B. Top Ten Most Popular Services

JOB1 - Initial Aggregation

For this task we have to aggregate the transactions occurring to see how much each address within the user space has been involved in.

Mapper :

The Transactions dataset has been used in this analysis. The **to_address** field of this dataset has been used as the key and the **value** field has been aggregated. The transaction dataset has been split with ’,’ . Try and catch has been used to only take the correct data. The value with ‘0’ has been removed. The address is passed as the key and the value has been passed for each address.

Combiner :

Combiner acts as a preliminary reducer. The code is similar to the reducer and has been used to increase the efficiency of the code.

Reducer :

The reducer takes the value for each key i.e addresses and aggregates the value and yields the results.

JOB ID : http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1574975221160_0076/

Below is the code that has been used for achieving the results.

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

class course(MRJob):

    def mapper(self, _, line):
        #the field has been splitted with the ','
        fields=line.split(",")
        try:
            if (len(fields)==7):
                to_address = fields[2]
                value = int(fields[3])
                if value==0:
                    pass
                else:
                    yield (to_address,value)
        except:
            pass
#reducer to sum the number of values per address
    def reducer(self, key, counts):
        yield (key,sum(counts))
#combiner to sum the number of values per address
    def combiner(self, key, counts):
        yield (key,sum(counts))

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

Job 2 - Joining transactions/contracts and filtering

For this task we have to join the **to_address** from the result we got from the first job and contracts dataset and filter out the missing contracts. This task has been achieved by using the repartition join.

Mapper :

The Contracts dataset and aggregated data has been used in this analysis. The **to_address** field of this dataset has been used as the key and the **value** field has been aggregated. The contracts dataset has been split with ’,’ and aggregated file from the output of first job with ‘\t’. Try and catch has been used to only take the correct data from the files. Join key and the Join value has been used and yielded. Join key has address which needs to be joined at the reducer. The address is passed as the key and the value has been passed for each address.

Reducer :

The reducer takes the value for each key i.e. addresses. A for loop is used to get the value from the two join key received from the mapper. After the value has been stored in the variables we use an id statement to check the value and yield only those which were mapped within the two tables and are not null.

Below is the code and Job ID.

JOB ID :
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1574975221160_0107/

```
from mrjob.job import MRJob

class repartition_stock_join(MRJob):

    def mapper(self, _, line):
        #the field has been splitted with the ',' and '\t' for two different files
        fields = line.split(",")
        field = line.split("\t")
        try:
            if(len(field)==2):
                join_key = field[0]
                join_key = join_key[1:-1]     #To remove the '""' from the field value
                join_value = int(field[1])
                yield (join_key, (join_value,1)) #Join key from the first file


            if(len(fields)==5):
                join_key = fields[0]
                join_value = fields[3]
                yield (join_key, (join_value,2)) #Join key from the second file
        except:
            pass

    def reducer(self, address, values):

        val = None
        exist = None
        #Matching the values from the mappers.
        for value in values:
            if value[1] == 1:
                val = value[0]
            elif value[1] == 2:
                exist = value[0]

        if val is not None and exist is not None : #The condition to only take the value which were present in both the tables
            yield(address,val)

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

Job 3 - Top Ten

For this task we have to get the to_address and values for top 10 from the result we got from the second job. This task has been achieved by using sorting in reverse order.

Mapper :

The value from the output of the second job is split with '\t' and the fields have been extracted. During yield the address and value has been sent to the reducer as a value and key has been kept None.

Reducer :

The values from the mapper have been sorted with the help of lambda function. During sorting the address is used as key and the values are sorted in reverse order. The for loop has been used to print the value for top 10 value as per address.

Below is the code and Job ID.

JOB ID :
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1574975221160_0684/

```
from mrjob.job import MRJob

class top_10(MRJob):

  def mapper(self, _, line):
    # try:
    fields=line.split('\t')    #the field has been splitted with the '\t'
    address = fields[0]
    address = address[1:-1]    #To remove the '""' from the field value
    value = int(fields[1])
    pair = (address,value)
    yield(None,pair)
    # except:
    #     pass

  def reducer(self, _,values):
      sorted_values = sorted(values, reverse=True,key=lambda l:l[1])  #lambda function to sort the values
      i=0
      for value in sorted_values:
          if i<10:
              yield(i,"{} - {}".format(value[0],value[1])) #yielding the values
              i+=1
          else:
            break
        # sorted_values = sorted_values[:10]
        # for value in sorted_values:
        #     address = value[0]
        #     value = value[1]
        #     yield(address,value)
if __name__ == '__main__':
    top_10.run()
```

Below is the result from the code for top 10:

Address_id	**Aggregated Value**
0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444	84155100809965865822726776
0xfa52274dd61e1643d2205169732f29114bc240b3	45787484483189352986478805
0x7727e5113d1d161373623e5f49fd568b4f543a9e	45620624001350712557268573
0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef	43170356092262468919298969
0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8	27068921582019542499882877
0xbfc39b6f805a9e40e77291aff27aee3c96915bdd	21104195138093660050000000
0xe94b04a0fed112f3664e45adb2b8915693dd5ff3	15562398956802112254719409
0xbb9bc244d798123fde783fcc1c72d3bb8c189413	11983608729202893846818681
0xabbb6bebfa05aa13e908eaa492bd7a8343760477	11706457177940895521770404
0x341e790174e3a4d35b65fdc067b6b5634a61caea	8379000751917755624057500

Part C. Data exploration

Miscellaneous Analysis:

2. Gas Guzzlers :

For the first part of the problem the gas price needs to be analyzed over time. For the sake of problem statement the time stamp and gas price over the period of the dataset has been used. Below is the code used to get the desired data.

JOB_ID :
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1574975221160_4846/

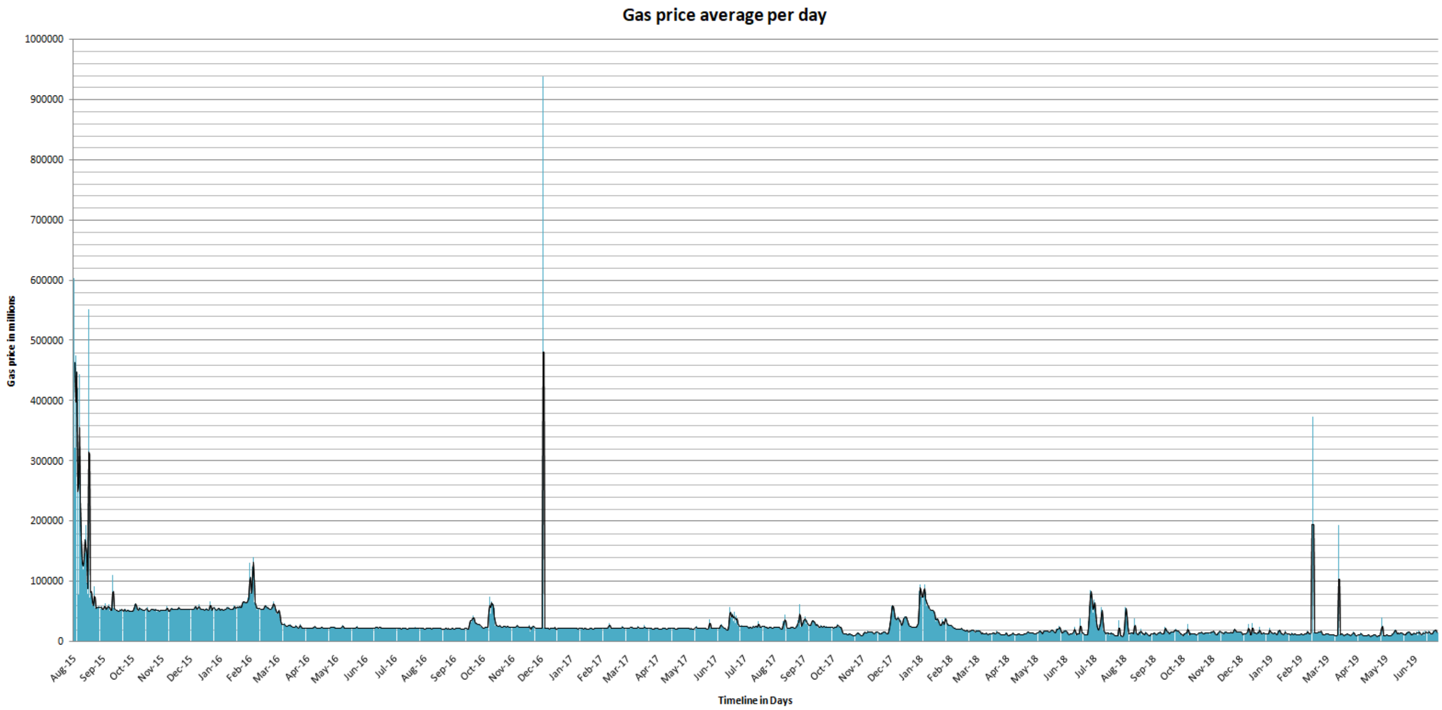
```
from mrjob.job import MRJob
import time
import statistics
class gas_guzzlers(MRJob):

    def mapper(self, _, line):
        try:
            fields = line.split(',')  #Line split
            if len(fields) == 7:
                date = int(fields[6])
                year = time.strftime('%Y-%m-%d', time.gmtime(date))  #epoch time change
                gas_price = float(int(fields[5]))
                yield (year,gas_price)
        except :
            pass

    def reducer(self, year, gas_price):
        yield (year,statistics.mean(gas_price))  #Average of gas price


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

Code Analysis:

Mapper : In the mapper the values have been split with ',' , year and gas price has been yielded. Date is taken as key and the gas price as value.

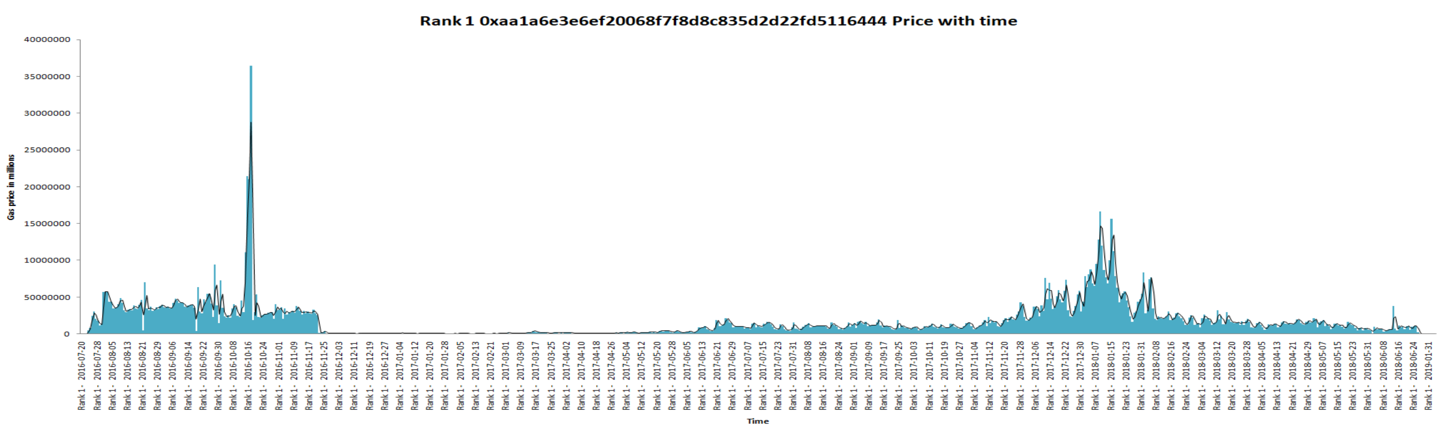
Reducer : In the reducer, the gas price has been averaged with the year as key.

The below graph has been plotted with the output data of the above program.

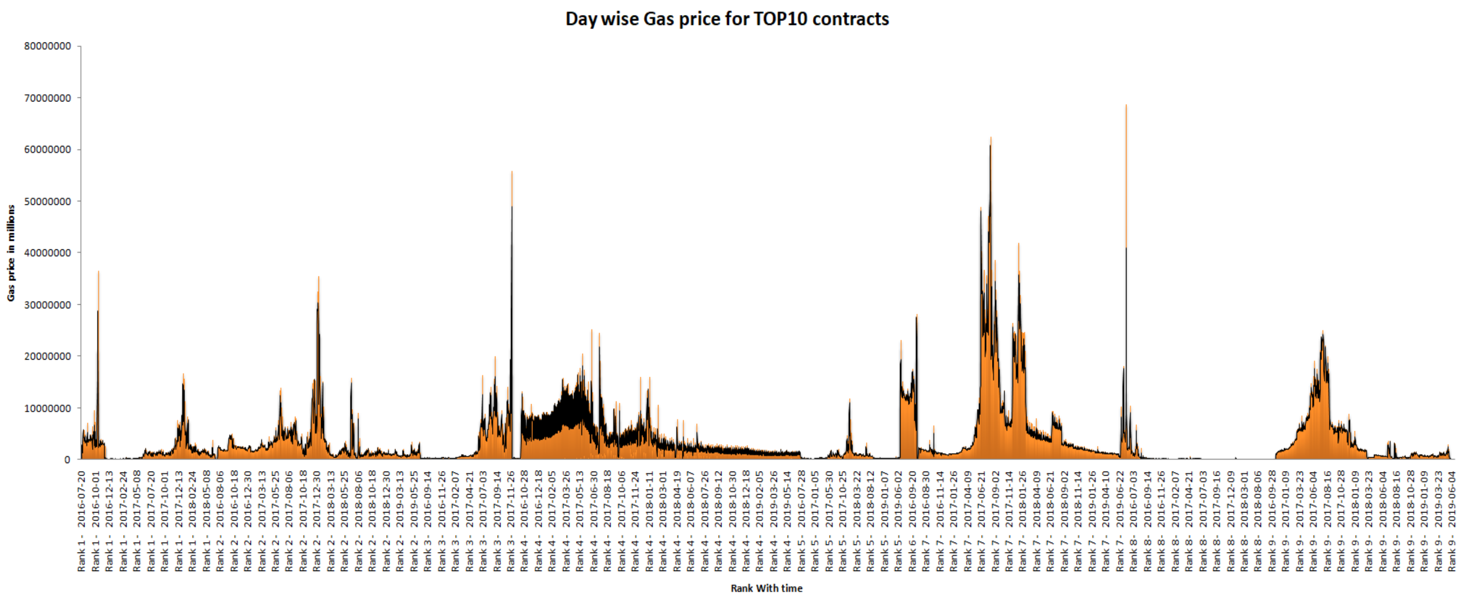


Graph Analysis : The distribution of Average Gas Price is assumed to be normal distribution.The graph has days in the X-axis and averaged gas_price per day in Y-axis. As can be seen from the graph the gas price started to depreciate from Aug-15 further going through the years. Spikes can be seen during the periods JAN-16 to Mar-16, DEC-17 to FEB-18 and JUL-18 to AUG-18. Certain large spikes can also be seen during this period i.e. in DEC-16, FEB-19,MAR-19 and AUG-15.

Graph for Rank 1 :



From the graph it can be seen that the Rank 1 has the similar gas price spikes as the average gas price as in 1 st graph in DEC-17 to FEB-18. Also the below graph has been plotted to see the gas prices of TOP10 ranks with the Gas price change over time. All the ranks seems to show some or more relation-ship between certain time periods and the gas prices as can be seen through the spikes.



For the second part of the GAS Guzzlers the Gas value has been averaged with the time. For this I have taken just the TOP 5 values from the JOB 2 result.

Code Analysis :

For the code I have used the replication join with two maps, one map has been used for taking the values of address and other has been joined with the address from the transactions table. From the transaction table epoch time, address and gas have been extracted. The matched address, year and gas have been yielded.

In the mapper the value of gas is averaged and yielded as per address id.

Below is the code for the task and the JOB ID for the same.

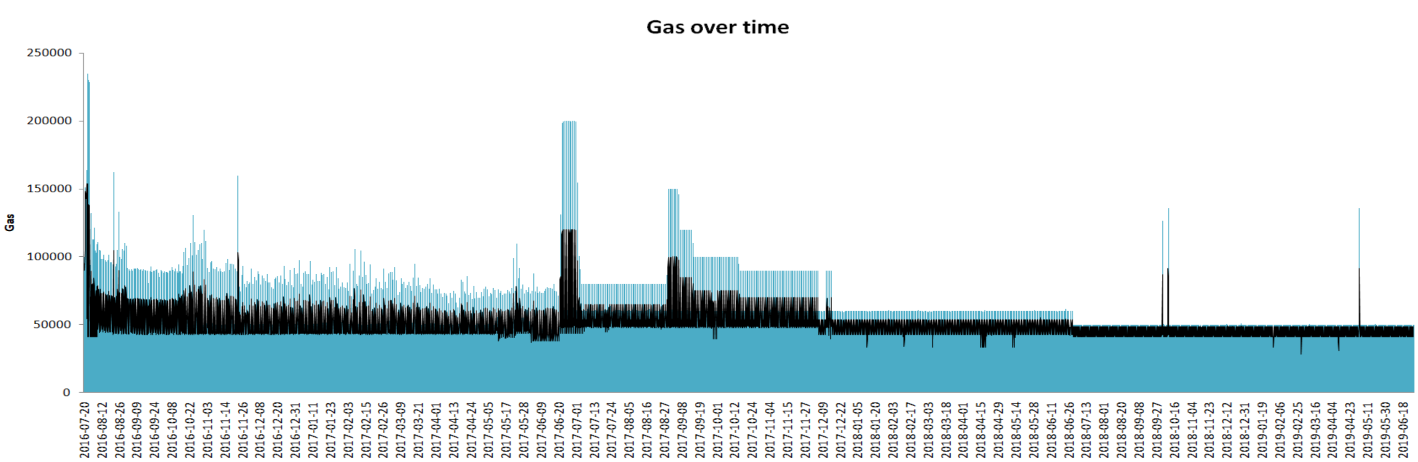
JOB ID

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1575381276332_3872/

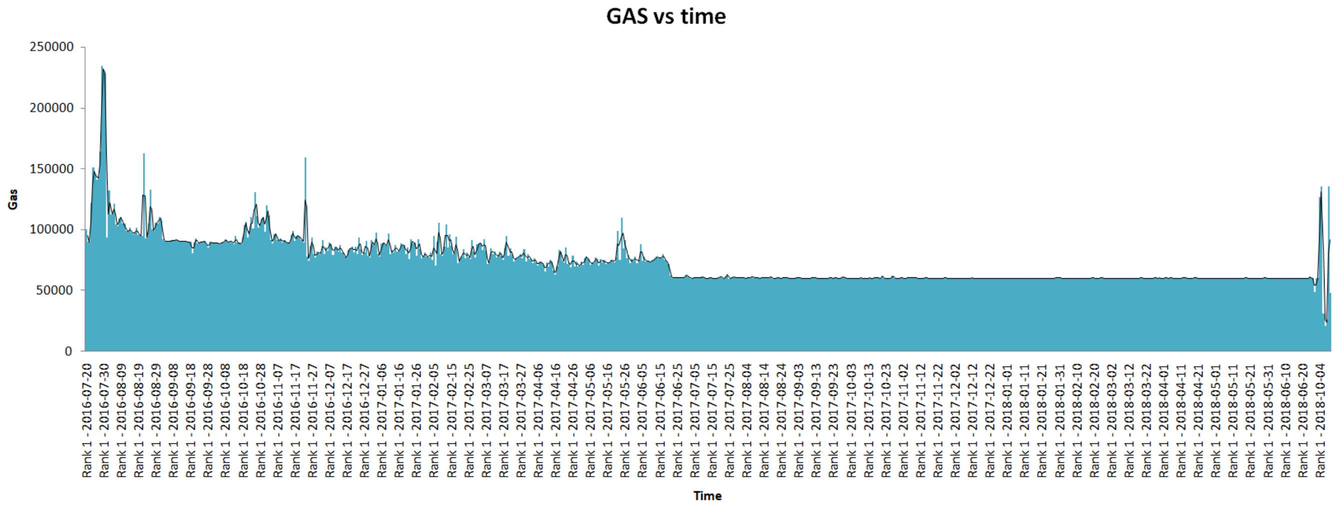
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1575381276332_3883/

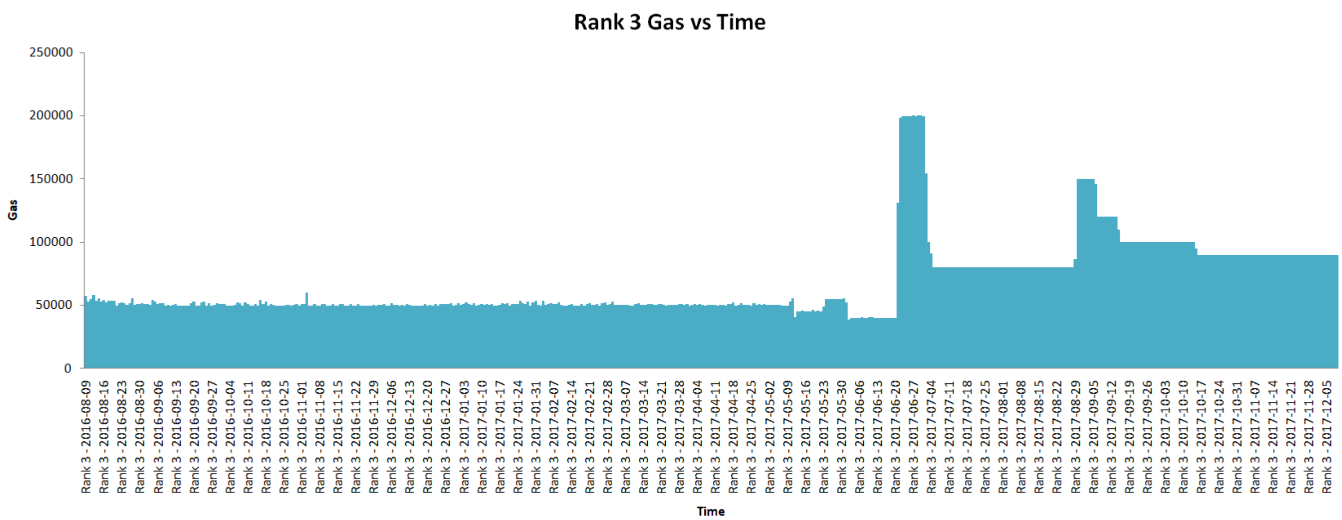
```python
from mrjob.job import MRJob
from mrjob.step import MRStep
import time
import statistics

class gas_guzzlers(MRJob):

    sector_table = {}

    def mapper_join_init(self):
        f = ['0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444','0xfa52274dd61e1643d2205169732f29114bc240b3','0x7727e5113d1d161373623e5f49fd568b4f543a9e',
        '0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef','0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8'] # Top 5 ranks
        for i in range(len(f)):
            key = f[i]
            value = None
            self.sector_table[key] = key

    def mapper_repl_join(self, _, line):
        # try:
        fields = line.split(',')
        if len(fields) == 7:
            address = fields[2]
            if address in self.sector_table:
                date = int(fields[6])
                year = time.strftime('%Y-%m-%d', time.gmtime(date))
                gas = float(int(fields[4]))
                key = (self.sector_table[address], year)  # The join has been done and the address has been mapped and sent as the key
                yield (key, gas)
        # except:
        #     pass


    def mapper_length(self, key, gas):
        yield(key, gas)


    def reducer_sum(self, key, gases):
        total = statistics.mean(gases)    #Mean has been found out for the gas
        yield (key, total)

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

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

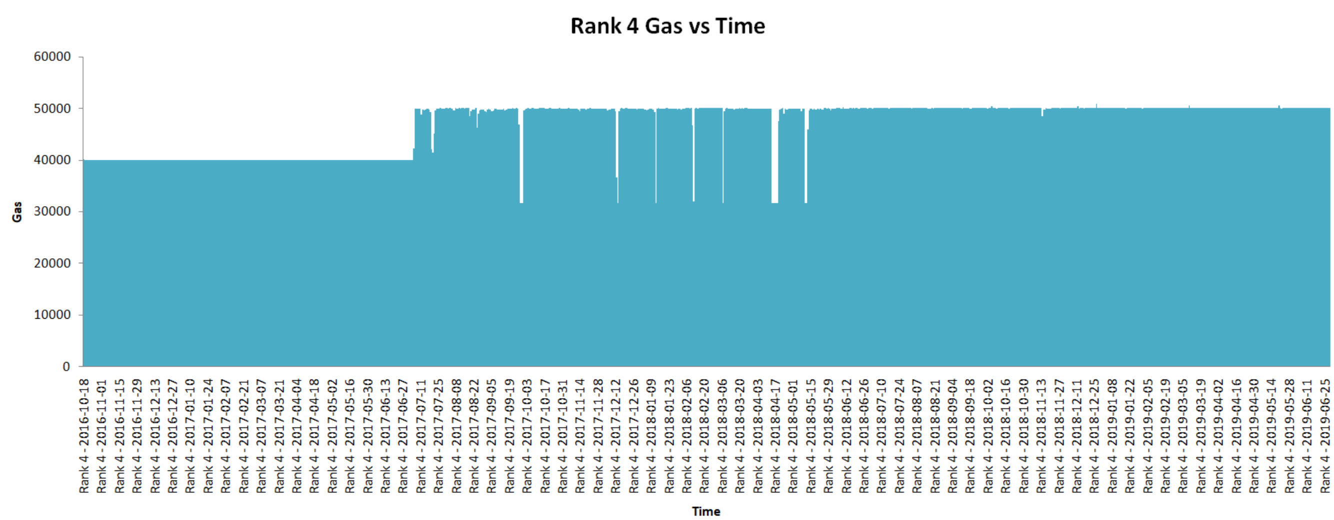
The graph has been plotted for the data between time and gas. We can see correlation between that the gas was changing over time. The contracts were changing throughout the period. The gas was fluctuating through the months of 2016 till 2017 but later in the years from 2018 we can see the values of gas becomes flatter which indicates the contracts were required gas on equilibrium or were less complicated . Also more the gas more the contracts are complicated and hence this can be easily seen from the spikes on certain intervals. So, yes the contracts were becoming complicated for some period of times during the tenure requiring more gas some times and required less gas for some periods.



Gas vs time graph has also been plotted individually for different ranks also to see the behavior how the gas has been changed over time . For some the gas can be seen changed over time and become less and flat. The contracts can be considered to show similar complication and gas requirements.







3. Comparative Evaluation:

My earlier JOB 2 code was written in MRJob, now the job has been rewritten in Spark. Comparison has been given in the trail for both the types.

Code Analysis :

Functions has been created to clean the values from the two dataset i.e. transactions and contracts, with try and catch . The data is filtered using filter function and the lambda function in mapper and reducer is used to store the data in variable. The data is then joined by the key value with the join function. Top 10 is then found out for the values with the takeOrdered function. With the help of the for loop the records are then printed out.

The pyspark code has faster and better run than the MRJob code. Comparison table has been created with average timings for multiple jobs runs with Job id's. For **MRJob** the value comes to be around **36 mins and 16 secs** and for **Spark** it comes out to be **2 mins and 6 secs**. The values itself shows how fast is the spark in comparison with MrJob. For this task spark seems to be appropriate one. The reason for this is as Spark can run in Hadoop clusters through YARN (Yet Another Resource Negotiator) and process everything in memory. The fact that it can run as a Hadoop module and as a standalone solution makes it tricky to directly compare and contrast with MapReduce. MapReduce is strictly a disk-based but Spark uses memory and can use disk for processing which makes spark a faster option over certain jobs. Also, as the normal functionality, the MRjob programs read from the cluster, performs operation and write back to the cluster and performs the same steps multiple times till the job is finished but in the spark programs these steps are only done once in the memory.

MrJob results aggregated

JOB ID	TASK 1 Time	TASK2 Time	TASK3 Time	Aggregate
application_1574975221160_0076 application_1574975221160_0107 application_1574975221160_0684	21 mins and 40sec	12 mins and 11 sec	36 sec	34 min and 30 sec
application_1574975221160_5462 application_1574975221160_5592 application_1574975221160_5799	26 mins and 46 sec	7 mins and 22 secs	35 sec	34 mins and 15 sec
application_1575381276332_0269 application_1574975221160_5622 application_1574975221160_5812	29 mins and 19 secs	9 mins and 21 sec	33 sec	39 mins and 13 sec
			Average ->	36 mins and 16 secs

Spark job times aggregated

JOB ID	TASK TIME
application_1574975221160_5300	3 mins and 21 secs
application_1574975221160_5398	1 mins and 50 secs
application_1574975221160_5411	2 mins and 13 secs
application_1574975221160_5423	2 mins and 1 sec
Average - >	2 mins and 6 secs

The code in spark is given as below.

```python
import pyspark
import re

#function to get the contracts dataset
def contracts(line):
    try:
        fields = line.split(',')
        if len(fields) != 5:
            return False
        return True
    except:
        return False

#function to get the transaction dataset
def transactions(line):
    try:
        fields = line.split(',')
        if len(fields) != 7:
            return False
        int(fields[3])
        return True
    except:
        return False


sc = pyspark.SparkContext()

transactionFileData = sc.textFile("/data/ethereum/transactions")

filteredTransactions = transactionFileData.filter(transactions)

mappedTransactions = filteredTransactions.map(lambda l: (l.split(",")[2], int(l.split(",")[3])))

reducedTransactions = mappedTransactions.reduceByKey(lambda a, b: a + b)

contactFileData = sc.textFile("/data/ethereum/contracts")

filteredContacts = contactFileData.filter(contracts)

mappedContracts = filteredContacts.map(lambda l: (l.split(",")[0],None))

joinTable = reducedTransactions.join(mappedContracts) #joining the tables

top10 = joinTable.takeOrdered(10, key=lambda x: -x[1][0]) #Ordering for top 10
for record in top10:
    print("{} : {}".format(record[0], record[1][0]))
```