BigDataCW_Ngoc_Bach_Nguyen

December 19, 2022

1 Part A - Time Analysis

Firstly, I retrived the transactions dataset as a text file and cleaned it up by filtering out the indivdual columns by splitting the text line at the comma i.e. *clean_lines = lines.filter(good_line)*.

I got the number of monthly transactions by mapping each transaction to a unique month in the dataset i.e. *transactionCount = clean_lines.map(lambda b: (time.strftime("%m %Y",time.gmtime(int(b.split(',')[11]))),1))*; which gives {*month* : 1} and reduce the number of key instances with the value using the sum i.e. *transactionCount = transactionCount.reduceByKey(operator.add)*, to get the total number of transactions per month.

I got the average monthly value of transactions by mapping each transaction value to a unique month in the dataset i.e. *transactionValues = clean_lines.map(lambda b: (time.strftime("%m %Y",time.gmtime(int(b.split(',')[11]))),int(b.split(',')[7])))*; which gives {*month* : *transaction*} and then using the *spark.sparkContext.broadcast(transactionValues.countByKey())* to broadcasts the common data (reusable) needed by tasks within each stage i.e. number of transactions' values for each month, and the broadcasted data is cache in serialized format and deserialized before executing each task. The average is then calculated by reducing the number of key instances with their associated values using the sum i.e. *transactionValues = transactionValues.reduceByKey(operator.add)*, to get the total value of all transactions per month and divding by total by number of transactions to get the monthly average value of transactions and mapping it back onto the transactionValues rdd i.e. *transactionValues = transactionValues.map(lambda x: (x[0], x[1]/transactionValRecords.value[x[0]]))*.

[ ]:

```python
#PART A Code, actual python file is in zip folder

import sys, string
import os
import socket
import time
import operator
import boto3
import json
from pyspark.sql import SparkSession
from datetime import datetime

if __name__ == "__main__":
```

```
spark = SparkSession\
    .builder\
    .appName("Ethereum")\
    .getOrCreate()

def good_line(line):
    try:
        fields = line.split(',')
        if len(fields)!=15:
            return False
        int(fields[11]) #typecase timestamp column to int
        int(fields[7]) #typecast value column to int
        return True
    except:
        return False

# shared read-only object bucket containing datasets
s3_data_repository_bucket = os.environ['DATA_REPOSITORY_BUCKET']

s3_endpoint_url = os.environ['S3_ENDPOINT_URL']+':'+os.environ['BUCKET_PORT']
s3_access_key_id = os.environ['AWS_ACCESS_KEY_ID']
s3_secret_access_key = os.environ['AWS_SECRET_ACCESS_KEY']
s3_bucket = os.environ['BUCKET_NAME']


hadoopConf = spark.sparkContext._jsc.hadoopConfiguration()
hadoopConf.set("fs.s3a.endpoint", s3_endpoint_url)
hadoopConf.set("fs.s3a.access.key", s3_access_key_id)
hadoopConf.set("fs.s3a.secret.key", s3_secret_access_key)
hadoopConf.set("fs.s3a.path.style.access", "true")
hadoopConf.set("fs.s3a.connection.ssl.enabled", "false")

lines = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-parvulus/transactions.csv")
clean_lines = lines.filter(good_line) #filter out individual columns in converted transactions text file by splitting at comma

#No. of Monthly Transactions
transactionCount = clean_lines.map(lambda b: (time.strftime("%m %Y",time.gmtime(int(b.split(',')[11]))),1)) #mapping to see how many transactions there are per month key
transactionCount = transactionCount.reduceByKey(operator.add) #reducing to get the total number of transactions per month key

#No. of Monthly Average Transactions
```

```
    transactionValues = clean_lines.map(lambda b: (time.strftime("%m %Y",time.gmtime(int(b.split(',')[11]))),int(b.split(',')[7]))) #(date, value) rdd

    #broadcasts the common data (reusable) needed by tasks within each stage i.e.  number of transactions' values for each date, result is a dictionary
    #broadcasted data is cache in serialized format and deserialized before executing each task.
    transactionValRecords = spark.sparkContext.broadcast(transactionValues.countByKey())


    transactionValues = transactionValues.reduceByKey(operator.add) #reducing to each date (key) and total transactions' values for each date (value)
    transactionValues = transactionValues.map(lambda x: (x[0], x[1]/transactionValRecords.value[x[0]])) #getting the average value of transactions per month


    now = datetime.now() # current date and time
    date_time = now.strftime("%d-%m-%Y_%H:%M:%S")

    my_bucket_resource = boto3.resource('s3',
            endpoint_url='http://' + s3_endpoint_url,
            aws_access_key_id=s3_access_key_id,
            aws_secret_access_key=s3_secret_access_key)

    my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' + date_time + '/monthly_transactions.txt')
    my_result_object.put(Body=json.dumps(transactionCount.take(100)))
    my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' + date_time + '/average_transactions.txt')
    my_result_object.put(Body=json.dumps(transactionValues.take(100)))

    spark.stop()
```

2 Part A Text Output

- Number of Monthly Transactions

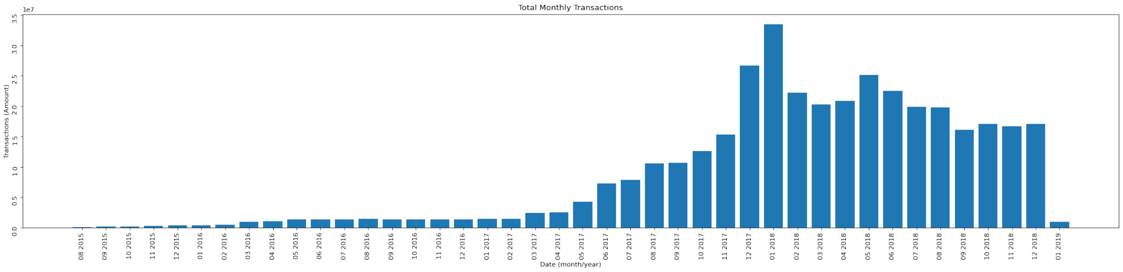
[[“10 2016”, 1329847], [“10 2015”, 205045], [“09 2015”, 173805], [“12 2016”, 1316131], [“04 2016”, 1023096], [“08 2017”, 10523178], [“12 2018”, 17107601], [“09 2018”, 16056742], [“05 2016”, 1346796], [“02 2017”, 1410048], [“02 2018”, 22231978], [“10 2017”, 12602063], [“07 2017”, 7835875], [“05 2018”, 25105717], [“08 2015”, 85609], [“11 2018”, 16713911], [“10 2018”, 17056926], [“07 2016”, 1356907], [“03 2017”, 2426471], [“11 2017”, 15292269], [“04 2017”, 2539966], [“06 2018”, 22471788], [“11 2015”, 234733], [“12 2017”, 26732085], [“08 2018”, 19842059], [“04 2018”, 20876642], [“09 2016”, 1387412], [“02 2016”, 520040], [“09 2017”, 10679242], [“06 2016”, 1351536], [“08 2016”, 1405743], [“06 2017”, 7244657], [“01 2017”, 1409664], [“07 2018”, 19937033], [“01 2019”, 1002431], [“11 2016”, 1301586], [“01 2018”, 33504270], [“03 2016”, 917170], [“12 2015”, 347092], [“05 2017”, 4245516], [“03 2018”, 20261862], [“01 2016”, 404816]]

- Average Value of Transactions per Month

[[“10 2016”, 3.2444426339709395e+19], [“10 2015”, 7.416931809333862e+19], [“09 2015”, 7.0464679457674535e+19], [“12 2016”, 6.146658677538069e+19], [“04 2016”, 2.2670632047054135e+19], [“08 2017”, 4.827395651885326e+19], [“12 2018”, 5.944894163484742e+18], [“09 2018”, 3.733481101982448e+18], [“05 2016”, 4.7046609524468195e+19], [“02 2017”, 5.558009016262998e+19], [“02 2018”, 6.230362795090817e+18], [“10 2017”, 2.6761515215631503e+19], [“07 2017”, 6.4981463792721e+19], [“05 2018”, 2.4981909136531256e+18], [“08 2015”, 4.8052118459597835e+20], [“11 2018”, 5.397909048901716e+18], [“10 2018”, 3.070402143025912e+18], [“07 2016”, 9.577823510582716e+19], [“03 2017”, 1.770215809422227e+20], [“11 2017”, 2.96411032747405e+19], [“04 2017”, 1.1135007462190998e+20], [“06 2018”, 2.8085234163056374e+18], [“11 2015”, 5.948474386250283e+19], [“12 2017”, 1.373122353832318e+19], [“08 2018”, 2.3989507981126523e+18], [“04 2018”, 2.449268234852099e+18], [“09 2016”, 3.2627612247557157e+19], [“02 2016”, 6.55476087599054e+19], [“09 2017”, 3.4372115150832353e+19], [“06 2016”, 3.0490334850065605e+19], [“08 2016”, 5.908198737290209e+19], [“06 2017”, 5.678772230936606e+19], [“01 2017”, 5.620285956535166e+19], [“07 2018”, 2.2749347554396106e+18], [“01 2019”, 4.2547896734505293e+18], [“11 2016”, 3.964431643835122e+19], [“01 2018”, 1.11645727207502e+19], [“03 2016”, 4.5853064127780815e+19], [“12 2015”, 2.6764096183940583e+19], [“05 2017”, 1.2484777365193273e+20], [“03 2018”, 2.728079891162356e+18], [“01 2016”, 6.106607047719591e+19]]

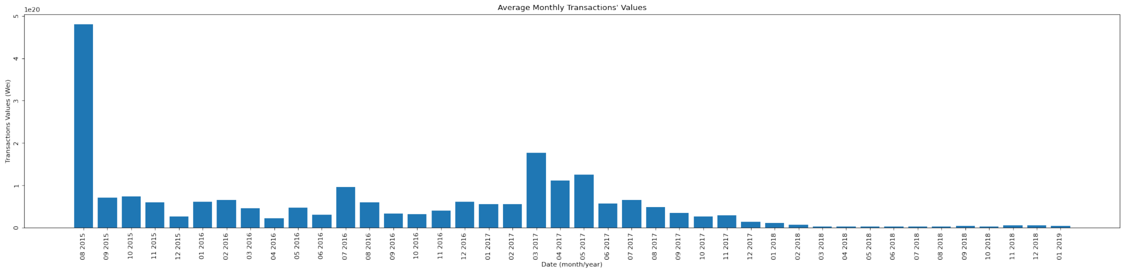
[2]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
parta1 = [["10 2016", 1329847], ["10 2015", 205045], ["09 2015", 173805], ["12␣
 ↪2016", 1316131], ["04 2016", 1023096], ["08 2017", 10523178], ["12 2018",␣
 ↪17107601], ["09 2018", 16056742], ["05 2016", 1346796], ["02 2017",␣
 ↪1410048], ["02 2018", 22231978], ["10 2017", 12602063], ["07 2017",␣
 ↪7835875], ["05 2018", 25105717], ["08 2015", 85609], ["11 2018", 16713911],␣
 ↪["10 2018", 17056926], ["07 2016", 1356907], ["03 2017", 2426471], ["11␣
 ↪2017", 15292269], ["04 2017", 2539966], ["06 2018", 22471788], ["11 2015",␣
 ↪234733], ["12 2017", 26732085], ["08 2018", 19842059], ["04 2018",␣
 ↪20876642], ["09 2016", 1387412], ["02 2016", 520040], ["09 2017", 10679242],␣
 ↪["06 2016", 1351536], ["08 2016", 1405743], ["06 2017", 7244657], ["01␣
 ↪2017", 1409664], ["07 2018", 19937033], ["01 2019", 1002431], ["11 2016",␣
 ↪1301586], ["01 2018", 33504270], ["03 2016", 917170], ["12 2015", 347092],␣
 ↪["05 2017", 4245516], ["03 2018", 20261862], ["01 2016", 404816]]
parta1_sort = (sorted(parta1,key=lambda x: x[0][3:]+x[0][:2]))
```

```
# Creating a 2 dimensional numpy array
data1a = (parta1_sort)


figure(figsize=(30, 6), dpi=80)
testList2 = [(elem1, elem2) for elem1, elem2 in data1a]
plt.bar(*zip(*testList2))
plt.xticks(rotation = 90)
plt.yticks(rotation = 90)
plt.title("Total Monthly Transactions")
plt.xlabel("Date (month/year)")
plt.ylabel("Transactions (Amount)")
plt.show()
```



[3]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
```

```
parta2 = [["10 2016", 3.2444426339709395e+19], ["10 2015", 7.
 ↪416931809333862e+19], ["09 2015", 7.0464679457674535e+19], ["12 2016", 6.
 ↪146658677538069e+19], ["04 2016", 2.2670632047054135e+19], ["08 2017", 4.
 ↪827395651885326e+19], ["12 2018", 5.944894163484742e+18], ["09 2018", 3.
 ↪733481101982448e+18], ["05 2016", 4.7046609524468195e+19], ["02 2017", 5.
 ↪558009016262998e+19], ["02 2018", 6.230362795090817e+18], ["10 2017", 2.
 ↪6761515215631503e+19], ["07 2017", 6.4981463792721e+19], ["05 2018", 2.
 ↪4981909136531256e+18], ["08 2015", 4.8052118459597835e+20], ["11 2018", 5.
 ↪397909048901716e+18], ["10 2018", 3.070402143025912e+18], ["07 2016", 9.
 ↪577823510582716e+19], ["03 2017", 1.770215809422227e+20], ["11 2017", 2.
 ↪96411032747405e+19], ["04 2017", 1.1135007462190998e+20], ["06 2018", 2.
 ↪8085234163056374e+18], ["11 2015", 5.948474386250283e+19], ["12 2017", 1.
 ↪373122353832318e+19], ["08 2018", 2.3989507981126523e+18], ["04 2018", 2.
 ↪449268234852099e+18], ["09 2016", 3.2627612247557157e+19], ["02 2016", 6.
 ↪55476087599054e+19], ["09 2017", 3.4372115150832353e+19], ["06 2016", 3.
 ↪0490334850065605e+19], ["08 2016", 5.908198737290209e+19], ["06 2017", 5.
 ↪678772230936606e+19], ["01 2017", 5.620285956535166e+19], ["07 2018", 2.
 ↪2749347554396106e+18], ["01 2019", 4.2547896734505293e+18], ["11 2016", 3.
 ↪964431643835122e+19], ["01 2018", 1.11645727207502e+19], ["03 2016", 4.
 ↪5853064127780815e+19], ["12 2015", 2.6764096183940583e+19], ["05 2017", 1.
 ↪2484777365193273e+20], ["03 2018", 2.728079891162356e+18], ["01 2016", 6.
 ↪106607047719591e+19]]
parta2_sort = (sorted(parta2,key=lambda x: x[0][3:]+x[0][:2]))

# Creating a 2 dimensional numpy array
dataa2 = (parta2_sort)



figure(figsize=(30, 6), dpi=80)
testList = [(elem1, elem2) for elem1, elem2 in dataa2]
plt.bar(*zip(*testList))
plt.xticks(rotation = 90)
plt.yticks(rotation = 90)
plt.title("Average Monthly Transactions' Values")
plt.xlabel("Date (month/year)")
plt.ylabel("Transactions Values (Wei)")
plt.show()
```



3 Part B - Top Ten Most Popular Services

I filtered out the transactions and contracts text files to ensure that the individual columns for each respective table are correctly formatted.

For transactions; mapping each value to a to_address key. For contracts; mapping each bytecode value to an address key, since the two tables need to have the same dimensions for the join function to work.

I then joined the two newly mapped transactions_features & contracts_features rdds: *joined_data = transactions_features.join(contracts_features)*. The join is the in-build inner join; treating transactions as the left-hand-side and contracts as the right-hand-side with the address being the primary key.

The Value and Bytecode columns are combined in a tuple after the join, so to retrieve value I needed to correctly retrieve its position in the tuple, then map each value to a an address key in the newly made joined_data rdd i.e. *joined_data = joined_data.map(lambda x: (x[0],x[1][0])) #(address, value)*.

The newly mapped joined_data rdd can then be reduced by totalling up the value for each address key i.e. *joined_data = joined_data.reduceByKey(operator.add)*.

The top 10 contracts by transactions' values can then be retrieved from the mapped&reduced joined rdd i.e. *top10=joined_data.takeOrdered(10, key=lambda x: -x[1])*.

[ ]:

```
#PART B Code, actual python file is in zip folder

import sys, string
import os
import socket
import time
import operator
from operator import add
import boto3
import json
from pyspark.sql import SparkSession
from datetime import datetime

if __name__ == "__main__":
    spark = SparkSession\
        .builder\
        .appName("Ethereum")\
        .getOrCreate()

    # shared read-only object bucket containing datasets
    s3_data_repository_bucket = os.environ['DATA_REPOSITORY_BUCKET']
```

```
  s3_endpoint_url = os.environ['S3_ENDPOINT_URL']+':'+os.
↪environ['BUCKET_PORT']
  s3_access_key_id = os.environ['AWS_ACCESS_KEY_ID']
  s3_secret_access_key = os.environ['AWS_SECRET_ACCESS_KEY']
  s3_bucket = os.environ['BUCKET_NAME']

  hadoopConf = spark.sparkContext._jsc.hadoopConfiguration()
  hadoopConf.set("fs.s3a.endpoint", s3_endpoint_url)
  hadoopConf.set("fs.s3a.access.key", s3_access_key_id)
  hadoopConf.set("fs.s3a.secret.key", s3_secret_access_key)
  hadoopConf.set("fs.s3a.path.style.access", "true")
  hadoopConf.set("fs.s3a.connection.ssl.enabled", "false")

  def good_line_transactions(line):
      try:
          fields = line.split(',')
          if len(fields)!=15:
              return False
          int(fields[7]) #need to typecast value column to int
          int(fields[11]) #legacy code, not needed for this part
          return True
      except:
          return False

  def good_line_contracts(line):
      try:
          fields = line.split(',')
          if len(fields)!=6:
              return False
          return True
      except:
          return False


  transactions = spark.sparkContext.textFile("s3a://" +␣
↪s3_data_repository_bucket + "/ECS765/ethereum-parvulus/transactions.csv")
  contracts = spark.sparkContext.textFile("s3a://" +␣
↪s3_data_repository_bucket + "/ECS765/ethereum-parvulus/contracts.csv")


  clean_transactions = transactions.filter(good_line_transactions) #filter␣
↪out individual columns in converted transactions text file by splitting at␣
↪comma
  clean_contracts = contracts.filter(good_line_contracts) #filter out␣
↪individual columns in converted contracts text file by splitting at comma
```

```
   transactions_features = clean_transactions.map(lambda x: (x.
↪split(",")[6],int(x.split(",")[7]))) # (to_address, value)
   contracts_features = clean_contracts.map(lambda x: (x.split(",")[0], x.
↪split(",")[1])) # (address, bytecode)

   joined_data = transactions_features.join(contracts_features) # (address ,␣
↪(value,bytecode))
   joined_data = joined_data.map(lambda x: (x[0],x[1][0])) #(address, value)
   joined_data = joined_data.reduceByKey(operator.add) #total up value for␣
↪each address from respective instances

   top10=joined_data.takeOrdered(10, key=lambda x: -x[1])

   now = datetime.now() # current date and time
   date_time = now.strftime("%d-%m-%Y_%H:%M:%S")

   my_bucket_resource = boto3.resource('s3',
           endpoint_url='http://' + s3_endpoint_url,
           aws_access_key_id=s3_access_key_id,
           aws_secret_access_key=s3_secret_access_key)

   my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' +␣
↪date_time + '/top10contracts.txt')
   my_result_object.put(Body=json.dumps(top10))

   spark.stop()
```

4 Part B Text Output

- Top 10 contracts by transactions' values

```
[[“0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444”,          84155363699941767867374641],
[“0x7727e5113d1d161373623e5f49fd568b4f543a9e”,           45627128512915344587749920],
[“0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef”,           42552989136413198919298969],
[“0xbfc39b6f805a9e40e77291aff27aee3c96915bdd”,           21104195138093660050000000],
[“0xe94b04a0fed112f3664e45adb2b8915693dd5ff3”,           15543077635263742254719409],
[“0xabbb6bebfa05aa13e908eaa492bd7a8343760477”,           10719485945628946136524680],
[“0x341e790174e3a4d35b65fdc067b6b5634a61caea”,            8379000751917755624057500],
[“0x58ae42a38d6b33a1e31492b60465fa80da595755”,            2902709187105736532863818],
[“0xc7c7f6660102e9a1fee1390df5c76ea5a5572ed3”,            1238086114520042000000000],
[“0xe28e72fcf78647adce1f1252f240bbfaebd63bcc”, 1172426432515823142714582]]
```

5 Part C - Top Ten Most Active Miners

Firstly, I retrived the blocks dataset as a text file and cleaned it up by filtering out the indivdual columns by splitting the text line at the comma i.e. *clean_blocks = blocks.filter(good_line_blocks)*.

I got the miner and their activity by mapping each block size to a miner key in the dataset i.e. *blocks_features = clean_blocks.map(lambda x: (x.split(“,”)[9],int(x.split(“,”)[12])))*; which gives $\{miner : size\}$ and reduce the number of key instances with the value using the sum i.e. *blocks_features = blocks_features.reduceByKey(operator.add)*, to get the total block size mined of each miner.

I then sorted by block sizes i.e. *blocks_features = blocks_features.sortBy(lambda sizes : sizes[1], ascending=False)* and then take the ordinal top 10 block sizes and the miner associated with them to get the top 10 most active miners i.e. *top10 = blocks_features.takeOrdered(10, key=lambda x: -x[1])*

[ ]:

```
#PART C Code, actual python file is in zip folder

import sys, string
import os
import socket
import time
import operator
from operator import add
import boto3
import json
from pyspark.sql import SparkSession
from datetime import datetime

if __name__ == "__main__":
    spark = SparkSession\
        .builder\
        .appName("Ethereum")\
        .getOrCreate()

    # shared read-only object bucket containing datasets
    s3_data_repository_bucket = os.environ['DATA_REPOSITORY_BUCKET']

    s3_endpoint_url = os.environ['S3_ENDPOINT_URL']+':'+os.
  ↪environ['BUCKET_PORT']
    s3_access_key_id = os.environ['AWS_ACCESS_KEY_ID']
    s3_secret_access_key = os.environ['AWS_SECRET_ACCESS_KEY']
    s3_bucket = os.environ['BUCKET_NAME']

    hadoopConf = spark.sparkContext._jsc.hadoopConfiguration()
    hadoopConf.set("fs.s3a.endpoint", s3_endpoint_url)
    hadoopConf.set("fs.s3a.access.key", s3_access_key_id)
    hadoopConf.set("fs.s3a.secret.key", s3_secret_access_key)
    hadoopConf.set("fs.s3a.path.style.access", "true")
    hadoopConf.set("fs.s3a.connection.ssl.enabled", "false")
```

```python
def good_line_blocks(line):
    try:
        fields = line.split(',')
        if len(fields)!=19:
            return False
        int(fields[12]) #typecast size column to int
        return True
    except:
        return False


blocks = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-parvulus/blocks.csv")

clean_blocks = blocks.filter(good_line_blocks)

blocks_features = clean_blocks.map(lambda x: (x.split(",")[9],int(x.split(",")[12]))) # (miner, size)

blocks_features = blocks_features.reduceByKey(operator.add) #total up value for each address from respective instances

blocks_features = blocks_features.sortBy(lambda sizes : sizes[1], ascending=False) #sort by block size

top10 = blocks_features.takeOrdered(10, key=lambda x: -x[1]) #take top 10 sizes

now = datetime.now() # current date and time
date_time = now.strftime("%d-%m-%Y_%H:%M:%S")

my_bucket_resource = boto3.resource('s3',
        endpoint_url='http://' + s3_endpoint_url,
        aws_access_key_id=s3_access_key_id,
        aws_secret_access_key=s3_secret_access_key)

my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' + date_time + '/top10sizes.txt')
my_result_object.put(Body=json.dumps(top10))

spark.stop()
```

6 Part C Text Output

- Top 10 Most Active Miners

[["0xea674fdde714fd979de3edf0f56aa9716b898ec8", 17453393724], ["0x829bd824b016326a401d083b33d092293333a8 12310472526], ["0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c", 8825710065], ["0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5", 8451574409], ["0xb2930b35844a230f00e51431acae96fe543a0347", 6614130661], ["0x2a65aca4d5fc5b5c859090a6c34d164135398226 3173096011], ["0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb", 1152847020], ["0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01", 1134151226], ["0x1e9939daaad6924ad004c2560e90804164900341 1080436358], ["0x61c808d82a3ac53231750dadc13c777b59310bd9", 692942577]]

7 Part D - Popular Scams

I filtered out the transactions and scams text files to ensure that the individual columns for each respective table are correctly formatted.

For transactions; mapping each value and date to an address key. For scams; mapping each id and category value to an address key, since the two tables need to have the same dimensions for the join function to work and I specifically need those fields for this task.

I then joined the two newly mapped transactions_features & scams_features rdds: *joined = transaction_features.join(scams_features)*. The join is the in-build inner join; treating transactions as the left-hand-side and scams as the right-hand-side with the address being the primary key.

I wanted to identify the most popular categories of scams as it will show me when the date they are active and all scams id are associated with a category, so it's important to know for later analysis.

The Category and Value columns are in separate tuples after the join, so to retrieve them, I needed to correctly retrieve their position in said tuples, then map each value to a a category key in the newly made scams_category rdd i.e. *scams_category = joined.map(lambda x: (x[1][1][1], x[1][0][1]))*.

The newly mapped scams_category rdd can then be reduced by totalling up the value for each category key i.e. *most_lucrative_scams = scams_category.reduceByKey(operator.add)*.

I then retrieved the Top 5 most lucrative categories but it turns out there were only three: Scamming, Phishing and Fake ICO i.e. *top5scamsCat = most_lucrative_scams.takeOrdered(5, key=lambda x: -x[1])*.

[ ]:

```python
#PART D - Scam Categories Code, actual python file is in zip folder

import sys, string
import os
import socket
import time
import operator
import boto3
import json
from pyspark.sql import SparkSession
from datetime import datetime

if __name__ == "__main__":

    spark = SparkSession\
        .builder\
        .appName("Ethereum")\
        .getOrCreate()

    def good_line_transactions(line):
        try:
            fields = line.split(',')
            if len(fields)!=15:
                return False
            int(fields[11])
            int(fields[8])
            return True
        except:
            return False

    def good_line_scams(line):
        try:
            fields = line.split(',')
            if len(fields)!=8:
                return False
            int(fields[0])
            return True
        except:
            return False

    # shared read-only object bucket containing datasets
    s3_data_repository_bucket = os.environ['DATA_REPOSITORY_BUCKET']

    s3_endpoint_url = os.environ['S3_ENDPOINT_URL']+':'+os.
 ↪environ['BUCKET_PORT']
    s3_access_key_id = os.environ['AWS_ACCESS_KEY_ID']
    s3_secret_access_key = os.environ['AWS_SECRET_ACCESS_KEY']
    s3_bucket = os.environ['BUCKET_NAME']


    hadoopConf = spark.sparkContext._jsc.hadoopConfiguration()
    hadoopConf.set("fs.s3a.endpoint", s3_endpoint_url)
    hadoopConf.set("fs.s3a.access.key", s3_access_key_id)
    hadoopConf.set("fs.s3a.secret.key", s3_secret_access_key)
    hadoopConf.set("fs.s3a.path.style.access", "true")
    hadoopConf.set("fs.s3a.connection.ssl.enabled", "false")

    transactions = spark.sparkContext.textFile("s3a://" +␣
↪s3_data_repository_bucket + "/ECS765/ethereum-parvulus/transactions.csv")
    scams = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket +␣
↪"/ECS765/ethereum-parvulus/scams.csv")

    clean_scams = scams.filter(good_line_scams)

    clean_transactions = transactions.filter(good_line_transactions)

    transaction_features = clean_transactions.map(lambda b: (b.
↪split(',')[6],(time.strftime("%m %y",time.gmtime(int(b.
↪split(',')[11]))),int(b.split(',')[7])))) #(address, (date, value))

    scams_features = clean_scams.map(lambda b: (b.split(',')[6],(int(b.
↪split(',')[0]),str(b.split(',')[4])))) #(address, (id, category))

    joined = transaction_features.join(scams_features) #(address, ((date,␣
↪value), (id, category)))

    #most lucrative form of scam id
    scams_category = joined.map(lambda x: (x[1][1][1], x[1][0][1])) #(category,␣
↪value)
    most_lucrative_scams = scams_category.reduceByKey(operator.add)
    top5scamsCat = most_lucrative_scams.takeOrdered(5, key=lambda x: -x[1])

    now = datetime.now() # current date and time
    date_time = now.strftime("%d-%m-%Y_%H:%M:%S")

    my_bucket_resource = boto3.resource('s3',
            endpoint_url='http://' + s3_endpoint_url,
            aws_access_key_id=s3_access_key_id,
            aws_secret_access_key=s3_secret_access_key)

    my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' +␣
↪date_time + '/most_lucrative_scam_categories.txt')
    my_result_object.put(Body=json.dumps(top5scamsCat))


    spark.stop()
```

8 Part D - Popular Scam Categories Text Output

- Most Lucrative Scams By Categories

[["Phishing", 43218561447276551309483], ["Scamming", 41626988623113580038109], ["Fake ICO", 1356457566889629979678]]

9 Part D - Popular Scams Continued

Following the logic of the popular scam categories code. I made the same steps to set up the joined rdd for clean_transactions and clean_scams i.e. *joined = transaction_features.join(scams_features) #(address, ((date, value), (id, category)))*. But, for the ID, I retrieved the ID column; instead of Category, along with the Value column i.e. *scams_id = joined.map(lambda x: (x[1][1][0], x[1][0][1])) #(id, value)*. I then got the top 5 lucrative scams by ID i.e. *top5scams = most_lucrative_scams.takeOrdered(5, key=lambda x: -x[1])*.

I also filter the ether recieved by scam categories to track the ether recieved over time (date, total_value) which shows when each scam type was most active over time i.e. *scamming = joined.filter(is_scamming).map(lambda x: (x[1][0][0], x[1][0][1])).reduceByKey(operator.add).sortByKey(ascending=True)*

[ ]:

```
#PART D - Scam ID and most lucrative scams over time Code, actual python file is in zip folder

import sys, string
import os
import socket
import time
import operator
import boto3
import json
from pyspark.sql import SparkSession
from datetime import datetime

if __name__ == "__main__":

    spark = SparkSession\
        .builder\
        .appName("Ethereum")\
        .getOrCreate()

    def good_line_transactions(line):
        try:
            fields = line.split(',')
            if len(fields)!=15:
                return False
            int(fields[11])
            int(fields[8])
            return True
        except:
```

```
        return False

def good_line_scams(line):
    try:
        fields = line.split(',')
        if len(fields)!=8:
            return False
        int(fields[0])
        return True
    except:
        return False

def is_scamming(x):
    if x[1][1][1] == 'Scamming':
        return True

def is_phishing(x):
    if x[1][1][1] == 'Phishing':
        return True

def is_fico(x):
    if x[1][1][1] == 'Fake ICO':
        return True


# shared read-only object bucket containing datasets
s3_data_repository_bucket = os.environ['DATA_REPOSITORY_BUCKET']

s3_endpoint_url = os.environ['S3_ENDPOINT_URL']+':'+os.
↪environ['BUCKET_PORT']
s3_access_key_id = os.environ['AWS_ACCESS_KEY_ID']
s3_secret_access_key = os.environ['AWS_SECRET_ACCESS_KEY']
s3_bucket = os.environ['BUCKET_NAME']


hadoopConf = spark.sparkContext._jsc.hadoopConfiguration()
hadoopConf.set("fs.s3a.endpoint", s3_endpoint_url)
hadoopConf.set("fs.s3a.access.key", s3_access_key_id)
hadoopConf.set("fs.s3a.secret.key", s3_secret_access_key)
hadoopConf.set("fs.s3a.path.style.access", "true")
hadoopConf.set("fs.s3a.connection.ssl.enabled", "false")

transactions = spark.sparkContext.textFile("s3a://" +␣
↪s3_data_repository_bucket + "/ECS765/ethereum-parvulus/transactions.csv")
scams = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket +␣
↪"/ECS765/ethereum-parvulus/scams.csv")
```

```
   clean_scams = scams.filter(good_line_scams)

   clean_transactions = transactions.filter(good_line_transactions)

   transaction_features = clean_transactions.map(lambda b: (b.
↪split(',')[6],(time.strftime("%m %y",time.gmtime(int(b.
↪split(',')[11]))),int(b.split(',')[7])))) #(address, (date, value))

   scams_features = clean_scams.map(lambda b: (b.split(',')[6],(int(b.
↪split(',')[0]),str(b.split(',')[4])))) #(address, (id, category))

   joined = transaction_features.join(scams_features) #(address, ((date,␣
↪value), (id, category)))

   #most lucrative form of scam id
   scams_category = joined.map(lambda x: (x[1][1][0], x[1][0][1])) #(id, value)
   most_lucrative_scams = scams_category.reduceByKey(operator.add)
   top5scams = most_lucrative_scams.takeOrdered(5, key=lambda x: -x[1])

   # most lucrative forms of scam vs time
   #(date, total_value)
   scamming = joined.filter(is_scamming).map(lambda x: (x[1][0][0],␣
↪x[1][0][1])).reduceByKey(operator.add).sortByKey(ascending=True)
   phishing = joined.filter(is_phishing).map(lambda x: (x[1][0][0],␣
↪x[1][0][1])).reduceByKey(operator.add).sortByKey(ascending=True)
   fico = joined.filter(is_fico).map(lambda x: (x[1][0][0], x[1][0][1])).
↪reduceByKey(operator.add).sortByKey(ascending=True)


   now = datetime.now() # current date and time
   date_time = now.strftime("%d-%m-%Y_%H:%M:%S")

   my_bucket_resource = boto3.resource('s3',
           endpoint_url='http://' + s3_endpoint_url,
           aws_access_key_id=s3_access_key_id,
           aws_secret_access_key=s3_secret_access_key)

   my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' +␣
↪date_time + '/most_lucrative_scams.txt')
   my_result_object.put(Body=json.dumps(top5scams))
   my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' +␣
↪date_time + '/scamming.txt')
   my_result_object.put(Body=json.dumps(scamming.take(100)))
   my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' +␣
↪date_time + '/phishing.txt')
   my_result_object.put(Body=json.dumps(phishing.take(100)))
```

```
    my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' + date_time + '/fico.txt')
    my_result_object.put(Body=json.dumps(fico.take(100)))


    spark.stop()
```

10 Part D - Popular Scam ID and Scam Categories Total Value Over Time Text Output

---

11 Most Lucrative Scams By ID:

12 [[5622, 16709083588073530571339], [2135, 6583972305381559340034], [90, 5972589629102419738989], [2258, 3462807524703738820904], [2137, 3389914242537183980016]]

- Phishing Total Value over Time

[[“01 18”, 2852623201117521082235], [“01 19”, 4124732020000000000], [“02 18”, 556277335842466025552], [“03 18”, 110503762814457062612], [“04 18”, 431378955798934750845], [“05 17”, 90000000000000000], [“05 18”, 981443006835040424695], [“06 17”, 1000000000000000000], [“06 18”, 999943232518418127096], [“07 17”, 10601485555862425267960], [“07 18”, 138035036261792915708], [“08 17”, 14175133698804715716729], [“08 18”, 48262224842858469476], [“09 17”, 3628698627630693984708], [“09 18”, 341374462555000000000], [“10 17”, 2771208122486755788873], [“10 18”, 34319384001457592063], [“11 17”, 3625307740195659707469], [“11 18”, 75302427910076440209], [“12 17”, 2004070215596173477729], [“12 18”, 145216740481604475524]]

- Scamming Total Value over Time

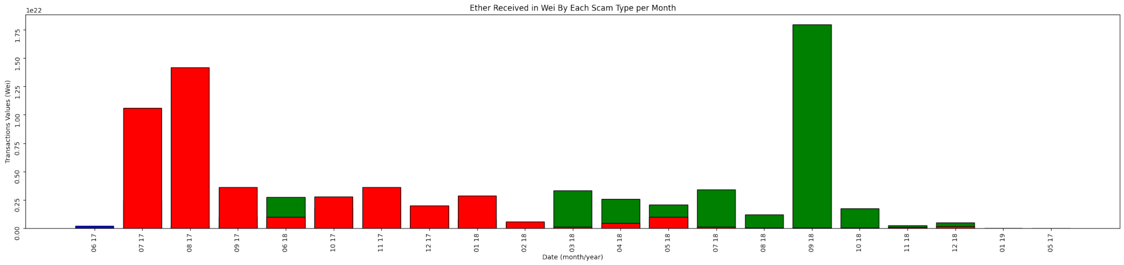
[[“01 18”, 803098320208667479640], [“01 19”, 1528806348292919943], [“02 18”, 548451069988353626717], [“03 18”, 3306005122269266940795], [“04 18”, 2591677970317916904218], [“05 18”, 2072461619819394441399], [“06 17”, 98784101200000000000], [“06 18”, 2744024628105125340064], [“07 17”, 2452981753628540121002], [“07 18”, 3392059628091102197748], [“08 17”, 30164742870000000000], [“08 18”, 1190915498037060761765], [“09 17”, 181866732896218700114], [“09 18”, 17950015894588455531420], [“10 17”, 1843586508663253070188], [“10 18”, 1759543527902562120422], [“11 17”, 3313865781457444576], [“11 18”, 237846121295971422203], [“12 17”, 28672302648918747084], [“12 18”, 478896099533022268811]]

- Fake ICO Total Value over Time

[[“06 17”, 182674023323763268000], [“06 18”, 12383182900000000000], [“07 17”, 16242199484949186112], [“08 17”, 181164662377136166221], [“09 17”, 975138363413781359345]]

[4]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

partdfico = [["06 17", 182674023323763268000], ["06 18", 123831829000000000],␣
 ↪["07 17", 162421994849186112], ["08 17", 181164662377136166221], ["09 17",␣
 ↪975138363413781359345]]
partdfico_sort = (sorted(partdfico,key=lambda x: x[0][3:]+x[0][:2]))

partdscamming = [["01 18", 803098320208667479640], ["01 19",␣
 ↪152880634829291994З], ["02 18", 548451069988353626717], ["03 18",␣
 ↪3306005122269266940795], ["04 18", 2591677970317916904218], ["05 18",␣
 ↪2072461619819394441399], ["06 17", 987841012000000000], ["06 18",␣
 ↪2744024628105125340064], ["07 17", 2452981753628540121002], ["07 18",␣
 ↪3392059628091102197748], ["08 17", 3016474287000000000], ["08 18",␣
 ↪1190915498037060761765], ["09 17", 181866732896218700114], ["09 18",␣
 ↪17950015894588455531420], ["10 17", 1843586508663253070188], ["10 18",␣
 ↪1759543527902562120422], ["11 17", 3313865781457444576], ["11 18",␣
 ↪237846121295971422203], ["12 17", 28672302648918747084], ["12 18",␣
 ↪478896099533022268811]]
partdscamming_sort = (sorted(partdscamming,key=lambda x: x[0][3:]+x[0][:2]))

partdphish = [["01 18", 2852623201117521082235], ["01 19",␣
 ↪412473202000000000], ["02 18", 556277335842466025552], ["03 18",␣
 ↪110503762814457062612], ["04 18", 431378955798934750845], ["05 17",␣
 ↪90000000000000000], ["05 18", 981443006835040424695], ["06 17",␣
 ↪1000000000000000000], ["06 18", 999943232518418127096], ["07 17",␣
 ↪10601485555862425267960], ["07 18", 138035036261792915708], ["08 17",␣
 ↪14175133698804715716729], ["08 18", 48262224842858469476], ["09 17",␣
 ↪3628698627630693984708], ["09 18", 34137446255500000000], ["10 17",␣
 ↪2771208122486755788873], ["10 18", 34319384001457592063], ["11 17",␣
 ↪3625307740195659707469], ["11 18", 75302427910076440209], ["12 17",␣
 ↪2004070215596173477729], ["12 18", 145216740481604475524]]
partdphish_sort = (sorted(partdphish,key=lambda x: x[0][3:]+x[0][:2]))

# Creating a 2 dimensional numpy array
datad1 = (partdfico_sort)
datad2 = (partdscamming_sort)
datad3 = (partdphish_sort)

figure(figsize=(30, 6), dpi=100)
testListfico = [(elem1, elem2) for elem1, elem2 in datad1]
testListscam = [(elem1, elem2) for elem1, elem2 in datad2]
testListphish = [(elem1, elem2) for elem1, elem2 in datad3]

plt.bar(*zip(*testListfico),color='blue',edgecolor='black')
plt.bar(*zip(*testListscam),color='green',edgecolor='black')
plt.bar(*zip(*testListphish),color='red',edgecolor='black')
plt.xticks(rotation = 90)
plt.yticks(rotation = 90)
plt.title("Ether Received in Wei By Each Scam Type per Month")
plt.xlabel("Date (month/year)")
plt.ylabel("Transactions Values (Wei)")
plt.show()
```



13 Note for Scams Bar Plot

Blue is Fake ICO, Red is Phishing and Green is Scamming

14 Part D - Gas Guzzlers 1

Firstly, I retrived the transactions dataset as a text file and cleaned it up by filtering out the indivdual columns by splitting the text line at the comma i.e. *clean_lines = lines.filter(good_line).*

I got the average gas price over time by mapping each gas price value to a date key i.e. *transactionValues = clean_lines.map(lambda b: (time.strftime(“%m %Y”,time.gmtime(int(b.split(‘,’)[11]))),int(b.split(‘,’)[9])))*, then I reduced the rdd by summing up the total gas price per unique date key instance i.e. *transactionValues = transactionValues.reduceByKey(operator.add)* and then I calculated the average gas price per month by dividing the total gas price by the number of months i.e. *transactionValues = transactionValues.map(lambda x: (x[0], x[1]/transactionValRecords.value[x[0]])) #(date, avg. gas_price)* (Similar approach to Part A).

[ ]:
```
#Part D - Gas Guzzlers 1, actual python file is in zip folder

import sys, string
import os
import socket
import time
import operator
import boto3
import json
from pyspark.sql import SparkSession
from datetime import datetime

if __name__ == "__main__":

    spark = SparkSession\
        .builder\
        .appName("Ethereum")\
        .getOrCreate()

    def good_line(line):
        try:
            fields = line.split(',')
            if len(fields)!=15:
                return False
            int(fields[11]) #typecast timestamp column to int
            int(fields[9]) #typecast gas price column to int
            return True
        except:
            return False

    # shared read-only object bucket containing datasets
    s3_data_repository_bucket = os.environ['DATA_REPOSITORY_BUCKET']

    s3_endpoint_url = os.environ['S3_ENDPOINT_URL']+':'+os.
 ↪environ['BUCKET_PORT']
    s3_access_key_id = os.environ['AWS_ACCESS_KEY_ID']
    s3_secret_access_key = os.environ['AWS_SECRET_ACCESS_KEY']
    s3_bucket = os.environ['BUCKET_NAME']


    hadoopConf = spark.sparkContext._jsc.hadoopConfiguration()
    hadoopConf.set("fs.s3a.endpoint", s3_endpoint_url)
    hadoopConf.set("fs.s3a.access.key", s3_access_key_id)
    hadoopConf.set("fs.s3a.secret.key", s3_secret_access_key)
    hadoopConf.set("fs.s3a.path.style.access", "true")
    hadoopConf.set("fs.s3a.connection.ssl.enabled", "false")

    lines = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket +␣
 ↪"/ECS765/ethereum-parvulus/transactions.csv")
    clean_lines = lines.filter(good_line) #clean transactions text file

    transactionValues = clean_lines.map(lambda b: (time.strftime("%m %Y",time.
 ↪gmtime(int(b.split(',')[11]))),int(b.split(',')[9]))) #(date, gas_price)
```

```
    transactionValRecords = spark.sparkContext.broadcast(transactionValues.
↪countByKey())
    transactionValues = transactionValues.reduceByKey(operator.add)
    transactionValues = transactionValues.map(lambda x: (x[0], x[1]/
↪transactionValRecords.value[x[0]])) #(date, avg. gas_price)


    now = datetime.now() # current date and time
    date_time = now.strftime("%d-%m-%Y_%H:%M:%S")

    my_bucket_resource = boto3.resource('s3',
            endpoint_url='http://' + s3_endpoint_url,
            aws_access_key_id=s3_access_key_id,
            aws_secret_access_key=s3_secret_access_key)

    my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' +␣
↪date_time + '/gas_guzzlers.txt')
    my_result_object.put(Body=json.dumps(transactionValues.take(100)))

    spark.stop()
```

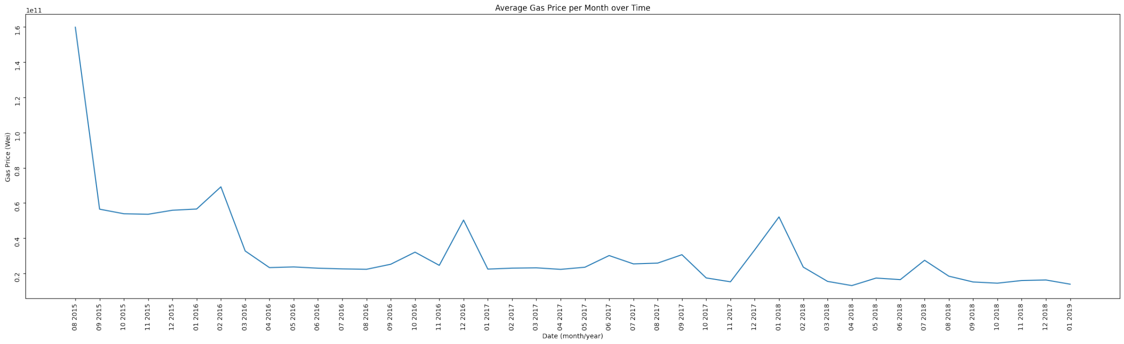
15 Part D - Gas Guzzlers 1 Text Output

- Average Gas Price Over Time

[[“10 2016”, 32112869584.914665], [“10 2015”, 53901692120.53661], [“09 2015”, 56511301521.033226], [“12 2016”, 50318068074.68128], [“04 2016”, 23361180502.721268], [“08 2017”, 25905774673.990257], [“12 2018”, 16338844844.014668], [“09 2018”, 15213870989.523378], [“05 2016”, 23746277028.263245], [“02 2017”, 23047230327.254303], [“02 2018”, 23636574203.828976], [“10 2017”, 17498286426.768925], [“07 2017”, 25460300456.232986], [“05 2018”, 17422505108.986416], [“08 2015”, 159744029578.03113], [“11 2018”, 16034859008.681648], [“10 2018”, 14526936383.350008], [“07 2016”, 22629542449.24175], [“03 2017”, 23232253600.81683], [“11 2017”, 15312465314.693544], [“04 2017”, 22355124545.395317], [“06 2018”, 16533308366.813036], [“11 2015”, 53607614201.796776], [“12 2017”, 33439362876.108334], [“08 2018”, 18483235826.894573], [“04 2018”, 13153739247.92998], [“09 2016”, 25270403393.626083], [“02 2016”, 69180681134.38849], [“09 2017”, 30675032016.988724], [“06 2016”, 23021251389.812134], [“08 2016”, 22396836435.95849], [“06 2017”, 30199442465.128727], [“01 2017”, 22507570807.719795], [“07 2018”, 27506077453.154327], [“01 2019”, 13954460713.077589], [“11 2016”, 24634294365.279953], [“01 2018”, 52106060636.84502], [“03 2016”, 32797039087.356667], [“12 2015”, 55899526672.35486], [“05 2017”, 23572314972.01526], [“03 2018”, 15549765961.743273], [“01 2016”, 56596270931.31685]]

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
```

```
partdgas1 = [["10 2016", 32112869584.914665], ["10 2015", 53901692120.53661],␣
 ↪["09 2015", 56511301521.033226], ["12 2016", 50318068074.68128], ["04 2016",␣
 ↪23361180502.721268], ["08 2017", 25905774673.990257], ["12 2018",␣
 ↪16338844844.014668], ["09 2018", 15213870989.523378], ["05 2016",␣
 ↪23746277028.263245], ["02 2017", 23047230327.254303], ["02 2018",␣
 ↪23636574203.828976], ["10 2017", 17498286426.768925], ["07 2017",␣
 ↪25460300456.232986], ["05 2018", 17422505108.986416], ["08 2015",␣
 ↪159744029578.03113], ["11 2018", 16034859008.681648], ["10 2018",␣
 ↪14526936383.350008], ["07 2016", 22629542449.24175], ["03 2017", 23232253600.
 ↪81683], ["11 2017", 15312465314.693544], ["04 2017", 22355124545.395317],␣
 ↪["06 2018", 16533308366.813036], ["11 2015", 53607614201.796776], ["12␣
 ↪2017", 33439362876.108334], ["08 2018", 18483235826.894573], ["04 2018",␣
 ↪13153739247.92998], ["09 2016", 25270403393.626083], ["02 2016", 69180681134.
 ↪38849], ["09 2017", 30675032016.988724], ["06 2016", 23021251389.812134],␣
 ↪["08 2016", 22396836435.95849], ["06 2017", 30199442465.128727], ["01 2017",␣
 ↪22507570807.719795], ["07 2018", 27506077453.154327], ["01 2019",␣
 ↪13954460713.077589], ["11 2016", 24634294365.279953], ["01 2018",␣
 ↪52106060636.84502], ["03 2016", 32797039087.356667], ["12 2015", 55899526672.
 ↪35486], ["05 2017", 23572314972.01526], ["03 2018", 15549765961.743273],␣
 ↪["01 2016", 56596270931.31685]]
partdgas1_sort = (sorted(partdgas1,key=lambda x: x[0][3:]+x[0][:2]))

# Creating a 2 dimensional numpy array
datagas1 = (partdgas1_sort)

figure(figsize=(30, 8), dpi=100)
testListgas1 = [(elem1, elem2) for elem1, elem2 in datagas1]

plt.plot(*zip(*testListgas1))
plt.xticks(rotation = 90)
plt.yticks(rotation = 90)
plt.title("Average Gas Price per Month over Time")
plt.xlabel("Date (month/year)")
plt.ylabel("Gas Price (Wei)")
plt.show()
```



16 Part D - Gas Guzzlers 2

I filtered out the transactions and contracts text files to ensure that the individual columns for each respective table are correctly formatted.

For transactions; mapping each gas value and date to an address key. For contracts; mapping each bytecode value to an address key, since the two tables need to have the same dimensions for the join function to work.

I then joined the two newly mapped transactions_features & contracts_features rdds: *joined = transaction_features.join(contract_features)*. The join is the in-build inner join; treating transactions as the left-hand-side and contracts as the right-hand-side with the address being the primary key.

The Date and Gas columns are combined in a tuple after the join, so to retrieve them, I needed to correctly retrieve their position in the tuple, then map each gas value to a a date key in the newly made monthly_average rdd i.e. *monthly_average = joined.map(lambda x: (x[1][0][0], x[1][0][1]))*.

The newly mapped monthly_average rdd can then be reduced by totalling up the gas value for each date key i.e. *monthly_average = monthly_average.reduceByKey(operator.add)*.

The average gas used per month is then calculated by dividing the total gas used by the number of instances that used gas within the month i.e. *monthly_average = monthly_average.map(lambda x: (x[0], x[1]/months.value[x[0]]))*.

[ ]:

```python
#Part D - Gas Guzzlers 2, actual python file is in zip folder

import sys, string
import os
import socket
import time
import operator
import boto3
import json
from pyspark.sql import SparkSession
from datetime import datetime

if __name__ == "__main__":

    spark = SparkSession\
        .builder\
        .appName("Ethereum")\
        .getOrCreate()

    def good_line_transactions(line):
        try:
```

```
        fields = line.split(',')
        if len(fields)!=15:
            return False
        int(fields[11]) #typecast timestamp column to int
        int(fields[8]) #typecast gas column to int
        return True
    except:
        return False

def good_line_contracts(line):
    try:
        fields = line.split(',')
        if len(fields)!=6:
            return False
        return True
    except:
        return False


# shared read-only object bucket containing datasets
s3_data_repository_bucket = os.environ['DATA_REPOSITORY_BUCKET']

s3_endpoint_url = os.environ['S3_ENDPOINT_URL']+':'+os.environ['BUCKET_PORT']
s3_access_key_id = os.environ['AWS_ACCESS_KEY_ID']
s3_secret_access_key = os.environ['AWS_SECRET_ACCESS_KEY']
s3_bucket = os.environ['BUCKET_NAME']


hadoopConf = spark.sparkContext._jsc.hadoopConfiguration()
hadoopConf.set("fs.s3a.endpoint", s3_endpoint_url)
hadoopConf.set("fs.s3a.access.key", s3_access_key_id)
hadoopConf.set("fs.s3a.secret.key", s3_secret_access_key)
hadoopConf.set("fs.s3a.path.style.access", "true")
hadoopConf.set("fs.s3a.connection.ssl.enabled", "false")

transactions = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-parvulus/transactions.csv")
contracts = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-parvulus/contracts.csv")

clean_transactions = transactions.filter(good_line_transactions)
clean_contracts = contracts.filter(good_line_contracts)

transaction_features = clean_transactions.map(lambda b: (b.split(',')[6],(time.strftime("%m %y",time.gmtime(int(b.split(',')[11]))),int(b.split(',')[8])))) #(address, (date, gas_used))
```

```
    contract_features = clean_contracts.map(lambda x: (x.split(",")[0], x.
↪split(",")[1])) # (address, byte)

    joined = transaction_features.join(contract_features) # (address, ((date,␣
↪gas_used), byte))

    monthly_average = joined.map(lambda x: (x[1][0][0], x[1][0][1]) ) # (date,␣
↪gas_used)
    months = spark.sparkContext.broadcast(monthly_average.countByKey()) #no. of␣
↪months
    monthly_average = monthly_average.reduceByKey(operator.add)
    monthly_average = monthly_average.map(lambda x: (x[0], x[1]/months.
↪value[x[0]]))  #(date, av. gas_used)

    now = datetime.now() # current date and time
    date_time = now.strftime("%d-%m-%Y_%H:%M:%S")

    my_bucket_resource = boto3.resource('s3',
            endpoint_url='http://' + s3_endpoint_url,
            aws_access_key_id=s3_access_key_id,
            aws_secret_access_key=s3_secret_access_key)

    my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' +␣
↪date_time + '/gas_monthly_average.txt')
    my_result_object.put(Body=json.dumps(monthly_average.take(100)))

    spark.stop()
```

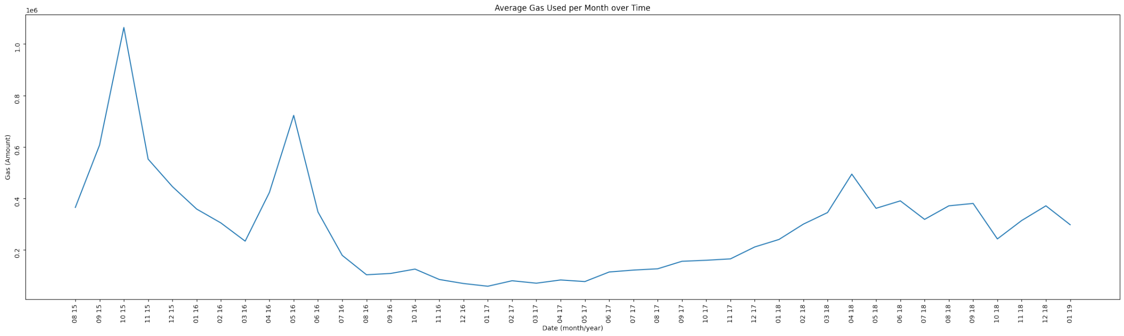
17 Part D - Gas Guzzlers 2 Text Output

- Average Gas Used Over Time

[[“09 17”, 155498.0710504653], [“11 17”, 164991.69973455978], [“05 16”, 722742.1068875446], [“10 15”, 1064341.6212], [“09 15”, 607479.3711739635], [“01 19”, 297607.5745047373], [“01 17”, 58527.96717527603], [“06 18”, 390420.2776759686], [“06 16”, 347263.67920171865], [“07 18”, 318457.6656414988], [“05 17”, 76781.97880585], [“01 16”, 358606.7810440953], [“04 16”, 423392.48398576514], [“04 17”, 82945.25166887816], [“08 15”, 364774.4365979381], [“06 17”, 113876.9744479187], [“03 16”, 233806.23642072213], [“02 17”, 79977.49780928274], [“08 17”, 126393.38194313418], [“10 16”, 125195.6470383587], [“03 18”, 344916.6728983044], [“07 16”, 178556.58677363582], [“11 15”, 552664.1394938199], [“02 16”, 304524.2311879603], [“04 18”, 494338.83781044686], [“10 18”, 242422.9136741674], [“12 18”, 371121.50413734367], [“12 17”, 211222.11535451622], [“10 17”, 159537.97692200154], [“08 16”, 103201.08623144105], [“09 18”, 380391.05840727483], [“08 18”, 371054.7681104046], [“09 16”, 108371.14506651415], [“11 16”, 84808.37901889467], [“02 18”, 299701.0380516722], [“05 18”, 361403.6319048684], [“11 18”, 313926.43661742256], [“12 16”, 69184.02698567968], [“03 17”, 70452.7594065135], [“12 15”, 445948.3533201189], [“07 17”, 121299.32492440357], [“01 18”, 240366.16341630253]]

[6]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

partdgas2 = [["09 17", 155498.0710504653], ["11 17", 164991.69973455978], ["05␣
 ↪16", 722742.1068875446], ["10 15", 1064341.6212], ["09 15", 607479.
 ↪3711739635], ["01 19", 297607.5745047373], ["01 17", 58527.96717527603],␣
 ↪["06 18", 390420.2776759686], ["06 16", 347263.67920171865], ["07 18",␣
 ↪318457.6656414988], ["05 17", 76781.97880585], ["01 16", 358606.7810440953],␣
 ↪["04 16", 423392.48398576514], ["04 17", 82945.25166887816], ["08 15",␣
 ↪364774.4365979381], ["06 17", 113876.9744479187], ["03 16", 233806.
 ↪23642072213], ["02 17", 79977.49780928274], ["08 17", 126393.38194313418],␣
 ↪["10 16", 125195.6470383587], ["03 18", 344916.6728983044], ["07 16", 178556.
 ↪58677363582], ["11 15", 552664.1394938199], ["02 16", 304524.2311879603],␣
 ↪["04 18", 494338.83781044686], ["10 18", 242422.9136741674], ["12 18",␣
 ↪371121.50413734367], ["12 17", 211222.11535451622], ["10 17", 159537.
 ↪97692200154], ["08 16", 103201.08623144105], ["09 18", 380391.05840727483],␣
 ↪["08 18", 371054.7681104046], ["09 16", 108371.14506651415], ["11 16", 84808.
 ↪37901889467], ["02 18", 299701.0380516722], ["05 18", 361403.6319048684],␣
 ↪["11 18", 313926.43661742256], ["12 16", 69184.02698567968], ["03 17", 70452.
 ↪7594065135], ["12 15", 445948.3533201189], ["07 17", 121299.32492440357],␣
 ↪["01 18", 240366.16341630253]]
partdgas2_sort = (sorted(partdgas2,key=lambda x: x[0][3:]+x[0][:2]))

# Creating a 2 dimensional numpy array
datagas2 = (partdgas2_sort)

figure(figsize=(30, 8), dpi=100)
testListgas2 = [(elem1, elem2) for elem1, elem2 in datagas2]

plt.plot(*zip(*testListgas2))
plt.xticks(rotation = 90)
plt.yticks(rotation = 90)
plt.title("Average Gas Used per Month over Time")
plt.xlabel("Date (month/year)")
plt.ylabel("Gas (Amount)")
plt.show()
```



18 Part D - Gas Guzzlers 3

For this part I did the exact same as Gas Guzzlers 2 but I filtered using the Top 10 contracts from Part B - Top Ten Most Popular Services when cleaning the transactions text file. To see if the popular contracts used more or less gas than the average.

[ ]:
```
#Part D - Gas Guzzlers 3, actual python file is in zip folder

import sys, string
import os
import socket
import time
import operator
import boto3
import json
from pyspark.sql import SparkSession
from datetime import datetime

if __name__ == "__main__":

    spark = SparkSession\
        .builder\
        .appName("Ethereum")\
        .getOrCreate()

    def good_line_transactions(line):
        try:
            fields = line.split(',')
            if len(fields)!=15:
                return False
            #Filter by top 10 popular contracts
            elif str(fields[6])=="0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444"␣
    ↪or str(fields[6])=="0x7727e5113d1d161373623e5f49fd568b4f543a9e" or␣
    ↪str(fields[6])=="0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef" or␣
    ↪str(fields[6])=="0xbfc39b6f805a9e40e77291aff27aee3c96915bdd" or␣
    ↪str(fields[6])=="0xe94b04a0fed112f3664e45adb2b8915693dd5ff3" or␣
    ↪str(fields[6])=="0xabbb6bebfa05aa13e908eaa492bd7a8343760477" or␣
    ↪str(fields[6])=="0x341e790174e3a4d35b65fdc067b6b5634a61caea" or␣
    ↪str(fields[6])=="0x58ae42a38d6b33a1e31492b60465fa80da595755" or␣
    ↪str(fields[6])=="0xc7c7f6660102e9a1fee1390df5c76ea5a5572ed3" or␣
    ↪str(fields[6])=="0xe28e72fcf78647adce1f1252f240bbfaebd63bcc":
                int(fields[11])
                int(fields[8])
                return True
        except:
            return False

    def good_line_contracts(line):
        try:
            fields = line.split(',')
            if len(fields)!=6:
                return False
            return True
        except:
            return False


    # shared read-only object bucket containing datasets
    s3_data_repository_bucket = os.environ['DATA_REPOSITORY_BUCKET']

    s3_endpoint_url = os.environ['S3_ENDPOINT_URL']+':'+os.
    ↪environ['BUCKET_PORT']
    s3_access_key_id = os.environ['AWS_ACCESS_KEY_ID']
    s3_secret_access_key = os.environ['AWS_SECRET_ACCESS_KEY']
    s3_bucket = os.environ['BUCKET_NAME']


    hadoopConf = spark.sparkContext._jsc.hadoopConfiguration()
    hadoopConf.set("fs.s3a.endpoint", s3_endpoint_url)
    hadoopConf.set("fs.s3a.access.key", s3_access_key_id)
    hadoopConf.set("fs.s3a.secret.key", s3_secret_access_key)
    hadoopConf.set("fs.s3a.path.style.access", "true")
    hadoopConf.set("fs.s3a.connection.ssl.enabled", "false")

    transactions = spark.sparkContext.textFile("s3a://" +␣
    ↪s3_data_repository_bucket + "/ECS765/ethereum-parvulus/transactions.csv")
    contracts = spark.sparkContext.textFile("s3a://" +␣
    ↪s3_data_repository_bucket + "/ECS765/ethereum-parvulus/contracts.csv")
```

```
    clean_transactions = transactions.filter(good_line_transactions)
    clean_contracts = contracts.filter(good_line_contracts)

    transaction_features = clean_transactions.map(lambda b: (b.
 ↪split(',')[6],(time.strftime("%m %y",time.gmtime(int(b.
 ↪split(',')[11]))),int(b.split(',')[8])))) #(address, (date, gas_used))
    contract_features = clean_contracts.map(lambda x: (x.split(",")[0], x.
 ↪split(",")[1])) # (address, byte)

    joined = transaction_features.join(contract_features) # (address, ((date,␣
 ↪gas_used), byte))

    monthly_average = joined.map(lambda x: (x[1][0][0], x[1][0][1]) ) # (date,␣
 ↪gas_used)
    months = spark.sparkContext.broadcast(monthly_average.countByKey())
    monthly_average = monthly_average.reduceByKey(operator.add)
    monthly_average = monthly_average.map(lambda x: (x[0], x[1]/months.
 ↪value[x[0]]))  #(date, av_gas_used)

    now = datetime.now() # current date and time
    date_time = now.strftime("%d-%m-%Y_%H:%M:%S")

    my_bucket_resource = boto3.resource('s3',
            endpoint_url='http://' + s3_endpoint_url,
            aws_access_key_id=s3_access_key_id,
            aws_secret_access_key=s3_secret_access_key)

    my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' +␣
 ↪date_time + '/gas_monthly_average.txt')
    my_result_object.put(Body=json.dumps(monthly_average.take(100)))

    spark.stop()
```

19 Part D - Gas Guzzlers 3 Text Output

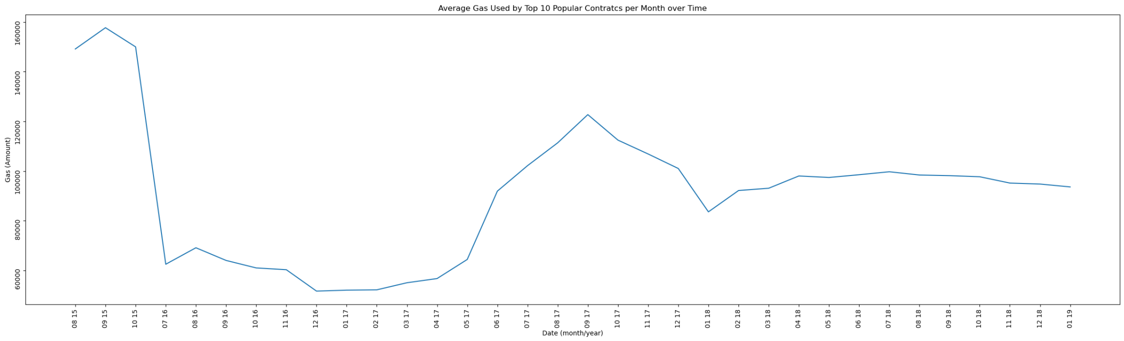
- Average Gas Used By Top 10 Contracts Over Time

[["09 17", 122729.77231868723], ["11 17", 106884.19092092774], ["10 15", 150000.0], ["09 15", 157720.7207207207], ["01 19", 93618.17253700868], ["01 17", 52050.74109700799], ["06 18", 98532.15100054273], ["07 18", 99717.70285112465], ["05 17", 64409.37275699514], ["04 17", 56727.26760775625], ["08 15", 149159.5754716981], ["06 17", 91929.18921087579], ["02 17", 52185.10474693569], ["08 17", 111371.84359086213], ["10 16", 61000.178378265606], ["03 18", 93087.44466493289], ["07 16", 62486.51043896785], ["04 18", 98030.44871758738], ["10 18", 97714.0116872388], ["12 18", 94776.95818876801], ["12 17", 101045.97104010075], ["10

17”, 112455.09551987745], [“08 16”, 69125.81623081565], [“09 18”, 98161.0080886826], [“08 18”, 98411.12662206819], [“09 16”, 64019.88633964143], [“11 16”, 60276.380738784654], [“02 18”, 92165.73941506378], [“05 18”, 97427.35923889813], [“11 18”, 95169.0485003718], [“12 16”, 51671.61439334538], [“03 17”, 55034.2956033601], [“07 17”, 102176.66018769318], [“01 18”, 83580.7206234139]]

[1]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

partdgas3 = [["09 17", 122729.77231868723], ["11 17", 106884.19092092774], ["10␣
 ↪15", 150000.0], ["09 15", 157720.7207207207], ["01 19", 93618.17253700868],␣
 ↪["01 17", 52050.74109700799], ["06 18", 98532.15100054273], ["07 18", 99717.
 ↪70285112465], ["05 17", 64409.37275699514], ["04 17", 56727.26760775625],␣
 ↪["08 15", 149159.5754716981], ["06 17", 91929.18921087579], ["02 17", 52185.
 ↪10474693569], ["08 17", 111371.84359086213], ["10 16", 61000.178378265606],␣
 ↪["03 18", 93087.44466493289], ["07 16", 62486.51043896785], ["04 18", 98030.
 ↪44871758738], ["10 18", 97714.0116872388], ["12 18", 94776.95818876801],␣
 ↪["12 17", 101045.97104010075], ["10 17", 112455.09551987745], ["08 16",␣
 ↪69125.81623081565], ["09 18", 98161.0080886826], ["08 18", 98411.
 ↪12662206819], ["09 16", 64019.88633964143], ["11 16", 60276.380738784654],␣
 ↪["02 18", 92165.73941506378], ["05 18", 97427.35923889813], ["11 18", 95169.
 ↪0485003718], ["12 16", 51671.61439334538], ["03 17", 55034.2956033601], ["07␣
 ↪17", 102176.66018769318], ["01 18", 83580.7206234139]]
partdgas3_sort = (sorted(partdgas3,key=lambda x: x[0][3:]+x[0][:2]))

# Creating a 2 dimensional numpy array
datagas3 = (partdgas3_sort)

figure(figsize=(30, 8), dpi=100)
testListgas3 = [(elem1, elem2) for elem1, elem2 in datagas3]

plt.plot(*zip(*testListgas3))
plt.xticks(rotation = 90)
plt.yticks(rotation = 90)
plt.title("Average Gas Used by Top 10 Popular Contratcs per Month over Time")
plt.xlabel("Date (month/year)")
plt.ylabel("Gas (Amount)")
plt.show()
```



20 Part D - Data Overhead

For Data Overhead, I just needed to know how many rows there were in the contracts dataset and how many characters for each of said rows in each unnecessary column. So, I cleaned the contracts textfile i.e. *clean_lines = lines.filter(good_line)*. Then I used two columns: sha3_uncles and logs_bloom for analysis (since the other columns in the example have the same character lengths as sha3_uncles). I then minus the first two characters (0x) from the total length as suggested in the problem definition and mapped the length to each row it appears in for sha3_uncles i.e. *dataSha = clean_lines.map(lambda x: (“sha3_uncles”, (len(x.split(“,”)[4])-2,1)))* and do the same for logs_bloom i.e. *dataLogs = clean_lines.map(lambda x: (“logs_bloom”, (len(x.split(“,”)[5])-2,1)))*.

I can then find out the total character length and total rows by summing up the tuple of each rdds respectively:

sha3_uncles: *dataSha = dataSha.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1]))*

logs_bloom: *dataLogs = dataLogs.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1]))*

[ ]:

```
#Part D - Data Overhead, actual python file is in zip folder

import sys, string
import os
import socket
import time
import operator
import boto3
import json
from pyspark.sql import SparkSession
from datetime import datetime

if __name__ == "__main__":

    spark = SparkSession\
        .builder\
        .appName("Ethereum")\
        .getOrCreate()

    def good_line(line):
        try:
            fields = line.split(',')
            if len(fields)!=19:
                return False
            return True
        except:
            return False

    # shared read-only object bucket containing datasets
    s3_data_repository_bucket = os.environ['DATA_REPOSITORY_BUCKET']

    s3_endpoint_url = os.environ['S3_ENDPOINT_URL']+':'+os.
↪environ['BUCKET_PORT']
    s3_access_key_id = os.environ['AWS_ACCESS_KEY_ID']
    s3_secret_access_key = os.environ['AWS_SECRET_ACCESS_KEY']
    s3_bucket = os.environ['BUCKET_NAME']


    hadoopConf = spark.sparkContext._jsc.hadoopConfiguration()
    hadoopConf.set("fs.s3a.endpoint", s3_endpoint_url)
    hadoopConf.set("fs.s3a.access.key", s3_access_key_id)
    hadoopConf.set("fs.s3a.secret.key", s3_secret_access_key)
    hadoopConf.set("fs.s3a.path.style.access", "true")
    hadoopConf.set("fs.s3a.connection.ssl.enabled", "false")

    lines = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket +␣
↪"/ECS765/ethereum-parvulus/blocks.csv")

    clean_lines = lines.filter(good_line)

    #-9 to value as sha3_uncles is 11 char then minus FIRST two. And -1 from␣
↪count because of the top row.
    dataSha = clean_lines.map(lambda x:  ("sha3_uncles", (len(x.
↪split(",")[4])-2,1)))
    print(dataSha.take(10)) #check string output in logs
    dataSha = dataSha.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1])) #reduce␣
↪tuple to total length and total instances in dataset i.e. rows

    #-8 to value as logs_bloom is 10 char then minus FIRST two. And -1 from␣
↪count because of the top row
    dataLogs = clean_lines.map(lambda x:  ("logs_bloom", (len(x.
↪split(",")[5])-2,1)))
```

```
    print(dataLogs.take(10)) #check string output in logs
    dataLogs = dataLogs.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1])) #reduce tuple to total length and total instances in dataset i.e. rows

    now = datetime.now() # current date and time
    date_time = now.strftime("%d-%m-%Y_%H:%M:%S")

    my_bucket_resource = boto3.resource('s3',
            endpoint_url='http://' + s3_endpoint_url,
            aws_access_key_id=s3_access_key_id,
            aws_secret_access_key=s3_secret_access_key)

    my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' + date_time + '/sha3_uncles.txt')
    my_result_object.put(Body=json.dumps(dataSha.take(100)))
    my_result_object = my_bucket_resource.Object(s3_bucket,'ethereum' + date_time + '/logs_bloom.txt')
    my_result_object.put(Body=json.dumps(dataLogs.take(100)))


    spark.stop()
```

21 Part D - Data Overhead Text Output

sha3_uncles Text output:

[[“sha3_uncles”, [448000073, 7000002]]]

Actual -> total characters length : 448000073 - 9 = 448000064, rows: 7000002 - 1 = 7000001

No. of bits for characters = 448000064 * 4 = 1792000256 bits

Check total no. of characters (sh3_uncles) = 7000001 * 64 = 448000064

logs_bloom Text output:

[[“logs_bloom”, [3584000520, 7000002]]]

Actual -> total characters length: 3584000520 - 8 = 3584000512, rows: 7000002 - 1 = 7000001

No. of bits for characters: 3584000512 * 4 = 14336002048 bits

Chars check: 7000001 * 512 (1 item in logs_blooms no. of chars) = 3584000512

22 Data Overhead Calculation

Sh3_uncles : 448000064 * 4 = 1792000256 bits

Logs_blooms: 3584000512 * 4 = 14336002048 bits

transactions_root: 448000064 * 4 = 1792000256 bits (same char length as sha3_uncles)

state_root: 448000064 * 4 = 1792000256 bits (same char length as sha3_uncles)

receipts_root: 448000064 * 4 = 1792000256 bits (same char length as sha3_uncles)

Total bits saved if the 5 columns are removed: 1792000256 * 4 + 14336002048 = 21504003072 bits = 2.688000384 GigaBytes