PART 1. TIME ANALYSIS

1. Explanation

Mapper()

The transactions.csv file serves as the input file. The Mapper function is called on every line of the file. the line is separated to get the value of time. For years and months, using (datetime.utcfromtimestamp) format to obtain them . The function combines year and month as key, emits all collection of <(year,month),1>.

Main code:
[

```
        fields = line.split(",")

        time = int(fields[2])

        year = datetime.utcfromtimestamp(time).year

        month =datetime.utcfromtimestamp(time).month

        yield ((year,month),1)
```

]

Reducer()

The Reducer () function is called on every key(year,month) and collected values. The function adds the value together according to every key and emits a partial result that is added to the output.

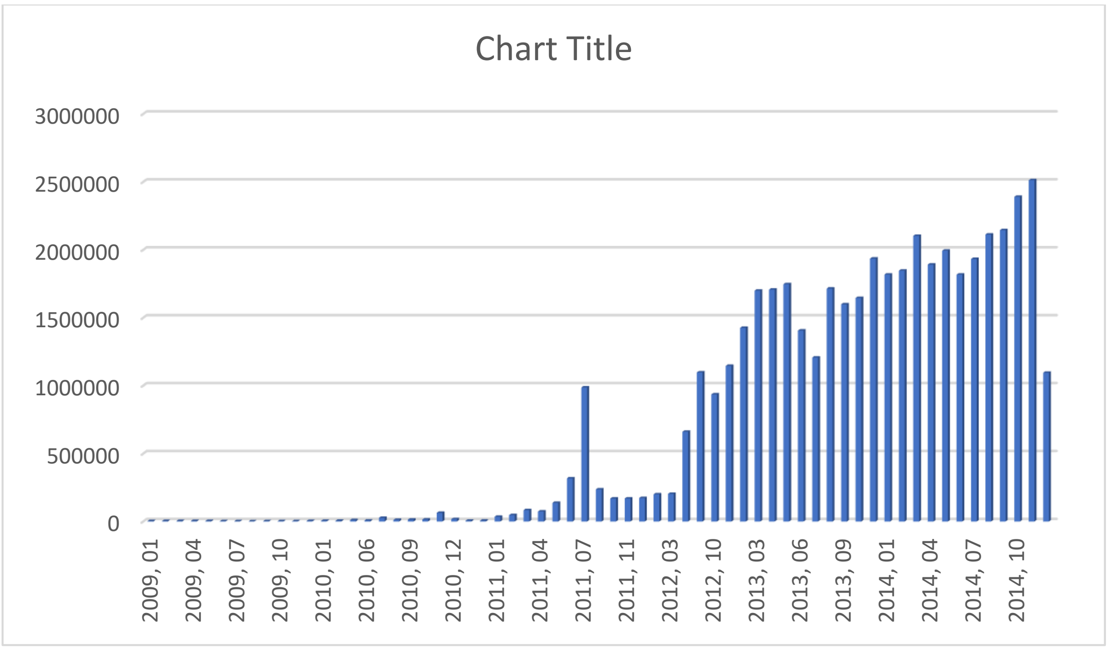
Main code :
[
```
        yield(key,sum(value))
```
]

2. Results:

2009, 01	2575
2009, 02	3417
2009, 03	3487
2009, 04	3459
2009, 05	3401
2009, 06	2244
2009, 07	1930
2009, 08	1570
2009, 09	2170
2009, 10	2139
2009, 11	2232

2009, 12	4084
2010, 01	5056
2010, 03	5398
2010, 04	9631
2010, 06	6678
2010, 07	26488
2010, 08	11968
2010, 09	13185
2010, 10	14386
2010, 11	63408
2010, 12	17142
2010,02	5751
2010,05	6212
2011, 01	34900
2011, 02	47168
2011, 03	83222
2011, 04	73936
2011, 05	136636
2011, 06	317482
2011, 07	986424
2011, 08	236300
2011, 10	168707
2011, 11	169012
2011, 12	172435
2012, 01	199876
2012, 03	203391
2012, 05	660620
2012, 08	1097435
2012, 10	935239
2012, 12	1145572
2013, 01	1425459
2013, 03	1699308
2013, 04	1707146
2013, 05	1746928
2013, 06	1405869
2013, 07	1206876
2013, 08	1714680
2013, 09	1597967
2013, 10	1645247
2013, 12	1935103
2014, 01	1817513
2014, 02	1846258
2014, 03	2102466
2014, 04	1890902
2014, 05	1993589
2014, 06	1817173

2014, 07	1932272
2014, 08	2112222
2014, 09	2144596
2014, 10	2390212
2014, 11	2512559
2014, 12	1094574



PART 2. TOP TEN DONORS

Obtain the top 10 donors over the whole dataset for the Wikileaks bitcoin address: `{1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v}`. Is there any information on who these wallets belong to? Can you work out how much was being donated if converted into pounds?

1. **Explanation**:

   Step1: Initial Filtering: job2_filter.py

   we need to get all the transaction Id from vout.csv related to address{1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v}. This is where we get a clue as to who transfer the money to the address. The output of step1 will be the one of inputs of second step.

   Main code:
   [

```
    key = "{1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v}"
```

```
      fields = line.split(",")

      if (fields[3] == key):
      print(f"{fields[0]},{fields[1]},{fields[2]},{fields[3]}")
```

]

Step2: First Join: job2_join1.py

The first join is to perform a replication join between a smaller version of the v_out file(out2.txt), which is from the output of step1, and big file vin.csv . The condition is [vout.hash==vin.txid] . the performance is to find the source of donation.

mapper_join_init():

The function opened out2.txt (the output of step1) to get joined key "hash". The key is saved in a list hash_table().

Main code:

[

```
       with open("out2.txt ") as f:

       for line in f:

            fields = line.split(",")

            hash = fields[0]

            self.hash_table.append(hash)
```

]

mapper_join1():

perform a replication join between a smaller version of the v_out file(job2_1.txt) and vin to get another file for preparing next join

Main code:

[

```
      def mapper_join1(self, _, line):

              fields = line.split(",")

              key = fields[0]

              if key in self.hash_table:

                    tx_hash = fields[1]

                    vout = fields[2]

            print(f"{fields[0]},{tx_hash},{vout}")
```

]

Step3: Second join:job2_join2.py
A file(out2.txt), which specifies the bitcoins is donated where they originated from, joined with the entire vout.csv. [vin.tx_hash=vout.hash , vin.vout = vout.n) this is aim to find how much bitcoins each donor donated ,and sort them for getting top10 donors. One mapper_init is called for prepare join data from the output of step2(out3.txt). A mapper is called for joining to get a collection of (public Key, value). There are two reducers are called. One is for sum the value by a public Key, another is for sort the top 10.

mapper_join_init():

The function is called for preparing joining data. The function opened the data from the output of step2(out3.txt), then Store the tx_hash and vout values from the out3.txt file into a dictionary "hash_table".

Main code:
[
```
      with open("out3.txt") as f:
      for line in f:
            fields = line.split(",")
            tx_hash = fields[1]
             vout = fields[2]
             self.hash_table[tx_hash]=vout
```
]

mapper_join2():

The function is called for matching the value of hash_table to the hash and n fields (respectively) within the full v_out file. When have a match on both, yield the publicKey (wallet) and value as a pair to the reducer

Main code :
[
```
       fields = line.split(",")
       hash = fields[0]
       publickey = fields[3]
       value = float(fields[1])
        n= fields[2]
        if (hash in self.hash_table and n in
self.hash_table[hash]) :
           yield(publickey,value)
```
]

reducer_count():

The **reducer** is called for summing the value in per publicKey.

Main code:

```
[
	yield(None,(feature,sum(value)))
]
```

reducer_sort ():

The **reducer** is called for sorting the value get top 10 publicKey.

Main code:

```
[

	sorted_values=sorted(values,reverse=True, key=lambda x: x[1])

	for i in range(10):

			yield(i+1,sorted_values[i])

]
```

Run steps:

```
[

	return [MRStep(mapper_init=self.mapper_join_init,

			mapper=self.mapper_join2,

			reducer=self.reducer_count),

		MRStep(reducer=self.reducer_sort)]

]
```

2. **Results:**

3324.41 pound=1 BTC(29/11/2018)

Public Key	Value(BTC)	Value(pounds)
17B6mtZr14VnCKaHkvzqpkuxMYKTvezDcp	46515.1894803	154,635,561
19TCgtx62HQmaaGy8WNhLvoLXLr7LvaDYn	5770.0	19181845.7
14dQGpcUhejZ6QhAQ9UGVh7an78xoDnfap	1931.482	6421038.
1LNWw6yCxkUmkhArb2Nf2MPw6vG7u5WG7q	1894.34	6297562.8394
1L8MdMLrgkCQJ1htiGRAcP11eJs662pYSS	806.134	2679919.93094
1ECHwzKtRebkymjSnRKLqhQPkHCdDn6NeK	648.52	2155946.3732
18pcznb96bbVE1mR7Di3hK7oWKsA1fDqhJ	637.04365574	2117533.112
19eXS2pE5f1yBggdwhPjauqCjS8YQCmnXa	576.835	1917636.04235

1B9q5KG69tzjhqq3WSz3H7PAxDVTAwNdbV	556.7	1850470.8
1AUGSxE5e8yPPLGd7BM2aUxfzbokT6ZYSq	500.0	1662000

I have tried to find the owner of address, but I did not find any sort of identification tied in to determine the location of a bitcoin address .

PART 3. Explore how much of BTC is used as a means of exchange (small transactions) as opposed to long term investment (Parked coins or large purchases)

1. **Explanation:**

According to my opinion, the exchange BTC is hard to catch at a moment, because it is always flowing. However, I can find the total amount of BTC , then take away the long term investment. (The amount of exchange = total bitcoin- large parked coins)

There are two ways to get the total BTC. The first one is very simple, just count the highest height of the block and multiply the rewards. For example, if the maximum height of the blocks is 334052 in block.csv, the total BTC=210000*50+124052*25=10500000+3101300=13601300. but it is not so accurate. A big amount of BTC was lost So I tried use another way to get the total BTC.

the total amount of BTC is the total mount BTC which left in the wallet. The two big files, vin.csv and vout.csv are joined together. The condition is vout. Hash !=vin. tx_hash , which means if hash in vout.csv never match tx_hash in vin.csv ,then amount BTC have been left. All the results generate a big file .This file ought to include all BTC left in wallet. Adding all the value to get the total amount of BTC .

In term of how much is a long-term investment, this assumption is that the investor hold a big amount of BTC for more than one year. For a large amount threshold, set to 1000.

2. **Results:**
I tried to join large file vin and vout. However, when Hadoop run up to 80% ,then it gave an error message:" beyond the physic memory". I guess the reason is the join is reducer join and too much shuffle to do,.So I used sample data to get the result, compared some information from data companies, and I think the percentage can reflect the reality state. The following result is from the sample data.

Total BTC	992627.6233620796
Large long-term investment	643967.28761751
Exchange BTC	348660.33574456966
Percentage of Large long-term investment	0.6487501178300483
Percentage of exchange BTC	0.3512498821699517

3. Implementation:

Step1:job3_step1.py

Join two big files vin and vout to produce another big file, the new big file include all BTC left in wallet.

mapper():

The function is called for join two big files vin.csv and vout.csv. which file the line comes from depends on the length of fields. The results generated by mapper follow the pattern:(hash or tx_hash,(file flag, count, value,publicKey),If the line is from vin.csv, the output is **(** tx_hash,(0,1,0,0)), if the line is from vou.csv, the output is (hash,(1,1,value,publicKey)).

Main code:

[

```
fields = line.split(",")
if len(fields)==4:
      hash=fields[0]
      value = fields[1]
      publicKey = fields[3]
      yield(hash,(1,1,value,publicKey))
if len(fields)==3:
      tx_hash=fields[1]
      yield(tx_hash,(0,1,0,0))
```

]

reducer()

the function is called for reducer join,getting the collection which satisify the condition:[vout.hash of is not in vin.tx_hash of vin]

Main code

"Repeat" is a variable to measure if vout.hash is in vin.tx_hash . if (repeat>1) means it is the BTC is spent, otherwise it is left in wallet.

[

```
repeat=0
for value in values:
    repeat=repeat+value[1]
    money=value[2]
    publicKey=value[3]
    if ((repeat==1) and (value[0]==1)):
```

```
                print(f"{key},{money},{publicKey}")
```

]

Step2:job3_step2.py :

There are have two mapper and two reducer. The first mapper and reducer is to get all BTC each wallet(publicKey).The second mapper and reducer is to filter the parked coin larger than 1000 and sum all ,yiled the last result.

The result is :(totalBTC,long term BTC,exchange BTC),(long term BTC / totalBTC, exchange BTC / totalBTC)

mapper_count():

the function is called for mapping all publicKey and the value which is not spent

main code:

[

```
      fields = lines.split(",")
      money=float(fields[1])
      publicKey=fields[2]
      yield(publicKey,money)
]
```

reducer_count():

The function is for adding all the BTC in each publicKey

main code:

```
[
      yield(key,sum(value))
]
```

mapper_total():

This function is called for preparing two values, one value is total BTC, another value is the big parked BTC. The threshold is 1000.

Main code:

```
[
      if (value>1000.0):
          yield(None,(value,value))
      else:
          yield(None,(value,0))


]
```

reducer_total():

this function is called for yielding four values:

the total BTC, The small exchange BTC, THE Percentage of small exchange BTC, The percentage of long-term invested BTC.

```
main code:

[

       totalsum=0

       largesum=0

       for value in values:

             totalsum=totalsum+value[0]

             largesum=largesum+value[1]

             yield((totalsum,largesum,totalsum-
largesum),(largesum/totalsum,totalsum-largesum/totalsum))

]

Steps:

[

       return [MRStep(mapper=self.mapper_count,

                    reducer=self.reducer_count),

              MRStep(mapper=self.mapper_total,

                    reducer=self.reducer_total)]

]
```

Step3:job3_step3.py

This step is to calculate long-term investment BTC more accurately, so the filter condition of investment years is added[transactions.time.year>1]. However, This step cannot run on the sample data, because transactionsSample is part of transactions.

mapper_join_init()

Preparing small data for join, the job3.txt from step1.

main code:

```
[

       with open("job3.txt") as f:

             for line in f:

               fields = line.split(",")

               if (len(fields)==3):

                 hash = fields[0]
```

```
                        money=float(fields[1])
                        publicKey=fields[2]
                        if (money>1000):
                           self.hash_table[hash]=(money,publicKey)
                           yield(None,money)
]
```

mapper_year()

from transactions.csv get the year to filter the investment years.

Main code:

```
[
        fields = lines.split(",")
         hash = fields[0]
         time = int(fields[2])
         year = datetime.utcfromtimestamp(time).year
         if ((hash in self.hash_table) and (year < 2014)):
            yield(None,self.hash_table[hash][0])
]
```

reducer_largeinvestment()

get investment BTC

main code:

```
[
          yield(None,sum(values))
]
```

PART 4. Identify different types of users e.g."whales" (volume trader/"hodler" of BTC), merchant accepting BTCs, or possibly criminal organizations.

1. **Explanation:**

assumptions about whales,merchant and possibly criminal organiztions:

Whales: the parked bitcoin is beyond 1000(totol bitcoin*60%/1000), and very few transactions(one or two). In my program, I changed the threshold be 4000, otherwise the mapper would not successes. The error is “beyond the physical memory”.

Merchant accepting BTCs: the number of transactions of bitcoin is at least larger than 100 and the exchange of BTC is a small amount, the parked coin in the address is less than 10. In my program, I used the numbers of transaction is from 20 to 40 for limit the size of data.

possibly criminal organizations: The address is only applicable to a large transaction in a short period of time (seconds). The purpose of address is only for launder money through bitcoin, so all transaction is mobile. The balance is 0. In most cases, when they stream a lot of bitcoins around, then they quickly give up the address, so there are only two transaction. One in and one out.

In order to verify the assumption, I used the following program to get the list of whales list, possible criminal organizations and merchant from vin and vout ,and search blockchain.com to see that were all correct. As for identifying merchant, due to the big data and limitation on my physical memory, so the condition is strict. Then there are 10 merchants in list.

2. result:

- Part of big **whale** list:(the full list is included in whales.txt, which identified 81 public Key,(these ID total hold 50%BTC). Of course, there are some addresses where the current balance is 0, because they drew their BTC after 09/2014.

Public Key	The parked money in our data	The parked money currently
15Z5YJaaNSxeynvr6uW6jQZLwq3n1Hu6RX	7641.06	7,941
12tkqA9xSoowkzoERHMWNKsTey55YEBqkv	12000	28,151
1PeizMg76Cf96nUQrYg8xuoZWLQozU5zGW	19400	19,414

- Part of possible **criminal** list:(the full list is included in criminal.txt)

Public Key	Flowing BTC	Transaction time
11CWSXL9ahce2c9md8yW3atPkt6VdGzJ4	11000	In: 2010-10-06 21:01:47 Out: 2010-10-09 04:37:57
12o29QSFhH3RrgfkGEmsd3akzeZCkvA3vU	26,550	In: 2012-03-24 05:12:11 Out: 2012-03-24 03:11:14
12zK3toTDKWVCFbXfrPx46KxgCeMN3AqS9	16,355	In: 2012-08-25 03:09:16 Out: 2012-08-25 04:27:19

- Part of **merchant** list (the full list is merchant.txt)

Public Key	The number of transactions	Total received(BTC)

12CDvRT86CRVbq6E9U95QFenHKWizXDaHj	60345	11,088.52651126 BTC
1dice2xkjAAiphomEJA5NoowpuJ18HT1s	75637	2,069.54874582 BTC
15ZY5nbr2SLtAP22La7323uTBEsM9XxfTZ	69652	3,055.99178556 BTC

3. implementation:

- **Identify possible criminal organization：**

Step1:Job4_criminal_step1.py:

Possible criminal and big whales have something in common: they both have a lot of BTC output. So firstly, I filter out the large output BTC from out.csv to get the file job4.txt. the threadhold is 4000, it is the most efficient to mapper and identify typical the features.

Main code:

```
[
        fields = line.split(",")
        hash=fields[0]
        value=float(fields[1])
         publicKey=fields[3]
        if (value > 4000):
                print(f"{hash},{value},{publicKey}")
]
```

Step2:Job4_criminal_step2.py

From the output of job4_criminal_step1.py, I can get a small file job4.txt. the file join job4.txt with vin.csv. The condition is vin.tx_hash=vout.hash. this means big amount BTC was flowing. Reducer was not included in Job4_criminal_step2.py,because there always was a physic memory limit error when running the reducer. So, another small file job4_2.txt is outputted for last join

Main code:

```
mapper_join_init （）
[
        with open("job4.txt") as f:
                for line in f:
                fields = line.split(",")
                 hash = fields[0]
                value = fields[1]
                publicKey = fields[2]
                self.hash_table[hash]=(publicKey,value)

]

mapper_criminal():

[

        fields = line.split(",")

         tx_hash = fields[1]

        if tx_hash in self.hash_table:

                publicKey = self.hash_table[tx_hash][0]

                value=self.hash_table[tx_hash][1]

                print(f"{hash},{value},{publicKey}")

]
```

Step3:Job4_criminal_step3.py

This is for shortlisting the possible criminal, so I supposed the transactions only happens twice. one input BTC and another output BTC. The input of the file is job4_2.txt which is from the output of step2.

Main code:

```
Mapper()

[

        fields = line.split(",")

         if len(fields)==3:

                hash=fields[0]

                value= float(fields[1])

                publicKey=fields[2]

                yield(publicKey,(value,1))

]

Reducer()

[

        for value in values:

                sumvalue = sumvalue+value[0]

                count = count+value[1]

        if (count==1):

                self.i = self.i+1
```

```
                        yield(feature,(self.i,sumvalue))
]
```

- **Identify the whales: Job4_whales.py**

To identify the whales, the basic idea is to remove all the possible criminal organisation publicKey from the big amount BTC transaction publicKey.(job4.txt-job4_2.txt)

mapper_join_init()

the function is called for prepare join data from job4_2.txt(possible criminal organization publicKey)

Main code:

```
[
            with open("job4_2.txt") as f:
             for line in f:
                  fields = line.split(",")
                 if len(fields)==3:
                      hash = fields[0]
                      value = fields[1]
                       publicKey = fields[2]
                       self.hash_table[hash]=(value,publicKey)
            ]
```

mapper_whales():

The function is aiming to getting all long-term big whales

Main code:

```
[
            fields = line.split(",")
            if len(fields)==3:
                        hash = fields[0]
                        value = fields[1]
                         publicKey = fields[2]
                        if hash not in self.hash_table:
                                    print(f"{hash},{value},{publicKey}"
```

]

- **Identify merchant: Job4_merchant.py**

The small BTC transaction (<1 BTC) is filtered from vout.csv . Then merge it to counting how many transactions happened in each publicKey. Only the publicKey , which satisfied with the two conditions: big number transaction times and small amount BTC exchange , is filter out. because the results is mapped very very slowly , so I narrow the range.

Mapper()

Filter out the small amount transactions:

Main code:

[

```
            fields = line.split(",")

            hash=fields[0]

            value=float(fields[1])



            publicKey=fields[3]

             if (value< 1):

                        yield(publicKey,(value,1))
```

]

Reducer()

The function is called to count the number of transactions.

Main code:

[

```
            for value in values:

            sumvalue = sumvalue + value[0]

          count = count + value[1]

           if ((count>20) and (count<40)):

                 yield(feature,(sumvalue,count))
```

]

PART 5 Ransomware often gets victims to pay via bitcoin. Find wallet IDs involved in such attacks and investigate how much money has been extorted. Explanation

When checking some features of Ransomware account that have been detected, some common point was found:

- in a short time, many transactions , and the amount is small.
- the total amount is small , mostly no more than 10 bitcoins, so the bitcoins would stream out quickly ,never left in the account.

Base on these features, I used vout .csv to filter out possible public Keys and find some that looks like scam. The process is similar to look for the merchant. First mapper all transaction value less than 1BTC from vout.csv, then reducer it . only the different way is the condition is different:

if ((count>20) and (count<40))and (sumvalue<20):

1. **Results:**
   Since the Ransomware started around 2012, then increase from 2017.It is difficult to find them due to our data range(2009-2014). So I only found one publicKey looks like scam.

   {17CzqBcwuB1aJTWEkJwhNYQuat3kKEoumE}"[0.92701449, 33]

2. **Implantation**

Mapper()

Main code:

```
[
        fields = line.split(",")
        hash=fields[0]
        value=float(fields[1])
        publicKey=fields[3]
        if (value< 1):
            yield(publicKey,(value,1))
]
```

Reducer()

Main code:

```
[
        sumvalue = 0
        count =0
        for value in values:
```

```
                    sumvalue = sumvalue + value[0]
                    count = count + value[1]
                     if ((count>20) and (count<40))and (sumvalue<20):
                          yield(feature,(sumvalue,count))
]
```